METODOLOGÍAS ÁGILES:

HERRAMIENTAS Y MODELO DE DESARROLLO PARA APLICACIONES JAVA EE COMO METODOLOGÍA EMPRESARIAL

Tesis Final de Máster

Jose Carlos Carvajal Riola

Septiembre 2008

Director: Dimas Cabre
Ponente: Dolors Costal
Septiembre 2008

Máster en Tecnologías de la Información - UPC - Barcelona

Agradecimientos

A todas las personas que han hecho posible este proyecto,
mi familia, mis compañeros de trabajo,
Dimas, Dolors y
Beatriz,
gracias por apoyarme y soportarme
en los momentos difíciles.

Any fool can make things bigger, more complex, and more violent. It takes a touch of genius - and a lot of courage - to move in the opposite direction" - Albert Einstein.

Dos por uno es dos, cinco, dos al cuadrado veinticinco, ctr+alt+supr. - Beatriz.

1. Indice de capítulos

2. Introducción

El primer capítulo de este documento introduce la situación característica en la que se desarrolla el proyecto, los motivos por los cuales se ha realizado, los objetivos marcados inicialmente y la estructura del documento. La utilización de una breve introducción como la que están leyendo ahora, va a ser una constante a lo largo del documento y que creemos que facilita al lector la identificación de los temas que más le interesan y la síntesis de los conceptos que se va a encontrar en el capítulo.

Contenidos de la sección

Orígenes y motivación

El proyecto surge con motivo de la finalización de los nuevos estudios denominados Master en Tecnologías de la Información (MTI). Se realiza en condición de becario y en colaboración con la empresa Everis, la cual ofertó un determinado número de proyectos para realizar con ellos y entre los cuales pudimos encontrar el que nos ocupa.
Las motivaciones personales que ocasionaron la elección de este proyecto y no cualquiera de los otros ofertados, vienen determinadas por una inquietud personal por el campo de la ingeniería del software y el mundo del desarrollo del software en general. Las metodologías ágiles están actualmente muy de moda y mis conocimientos sobre ellas eran escasos, sentía curiosidad por profundizar en un campo concreto del cual había visto muy poco en mis estudios universitarios. Otro factor que tuve en cuenta a la hora de escoger este proyecto, fue la posibilidad de desarrollarlo en una importante consultoría que me brindaba la posibilidad de ver, en primera persona, como funcionaba por dentro y que es lo que realmente hacían, que proyectos desarrollan, como se organizan, como trabajan y en definitiva, experimentar como es una consultoría desde dentro.

Objetivos del proyecto

Los objetivos del proyecto han ido evolucionando a lo largo de su desarrollo, al mismo tiempo que tomábamos conciencia de trabajo real que tenían las tareas que estábamos realizando. Desde el principio mi director de proyecto en la empresa ha propuesto ideas y objetivos para el proyecto, quizás hasta con demasiado entusiasmo, entusiasmo que me contagió y provocó que generásemos una lista de objetivos inicial bastante importante. A medida que avanzábamos en el proyecto, nos dimos cuenta que el proyecto era demasiado ambicioso para realizarlo en el tiempo limitado del cual disponíamos y empezamos a recortar objetivos que no eran viables temporalmente.

A continuación detallo el conjunto de objetivos que inicialmente se establecieron, pero que advertimos, muestran objetivos que finalmente no se han realizado:

1. Entender y documentar las características fundamentales de los proyectos software con Java EE.
2. Seleccionar un número de metodologías ágiles para su estudio.
3. Identificar un conjunto de criterios para la comparación de las metodologías.
4. Realizar comparativa del conjunto de metodologías ágiles seleccionadas.
5. Poner en práctica las metodologías ágiles en un proyecto Java EE sencillo.
6. Identificar herramientas para desarrollar proyectos software con Java EE y Metodologías ágiles.
7. Seleccionar herramientas para realizar comparativa a través de prueba de concepto.
8. Identificar conjunto de criterios para la comparación de las metodologías.
9. Identificar problemas de las metodologías estudiadas y buscar soluciones.
10. Estudio metodología empresarial COM y como las metodologías ágiles pueden mejorarla.

Desde el principio hemos tenido claro que uno de los principales objetivos era el estudio, análisis y comparativa de las metodologías ágiles, pero siempre desde la perspectiva de los proyectos empresariales desarrollados con Java EE. Es por esto que los objetivos uno, dos, tres y cuatro se han mantenido prácticamente invariables a lo largo de todo el proyecto. Tampoco ha sufrido ningún cambio el objetivo de realizar un caso práctico de aplicación de metodologías ágiles, en un proyecto con tecnologías Java EE, pero los mayores problemas los hemos encontrado a partir de este punto, ya que la elaboración de una comparativa de herramientas, nos condicionaba a realizar una prueba del mismo producto y un incremento temporal muy importante y del cual no disponíamos. Nos hemos encontrado con un problema similar a la hora de realizar el estudio de la metodología empresarial COM, no es una metodología simple y a pesar de iniciar su estudio, observamos que no disponíamos

de tiempo suficiente como para poder algo útil a esta metodología y preferimos centrar nuestros esfuerzos en otros puntos del proyecto.
De este modo, los objetivos que sí podemos encontrar desarrollados en este proyecto son los siguientes y que prácticamente se corresponden con cada uno de los capítulos de este documento:

1. *Entender y documentar las características fundamentales de los proyectos software realizados con JEE.*

Este objetivo nos ayuda a alcanzar un mayor conocimiento de la tecnología Java Enterprise Edition y de los tipos de proyectos que se pueden realizar con ella. Es necesario desarrollar este objetivo en las primeras etapas del proyecto, ya que los conocimiento adquiridos en su desarrollo nos serán requeridos para a consecución de otros objetivos. La identificación de las diferentes aplicaciones empresariales y su posterior caracterización son sub-objetivos necesarios para alcanzar el fin de este punto.

2. *Seleccionar un número de metodologías ágiles para su estudio.*

Debido al gran número de metodologías ágiles que actualmente existen (hemos encontrado más de 16 metodologías ágiles), es necesario restringir el ámbito de estudio, ya no tan solo por limitaciones temporales, sino también por la complejidad que supone trabajar con un número muy elevado de metodologías. La selección de metodología supone para el proyecto un tarea crítica y que debe realizarse justificadamente, bajos unos criterios objetivos.

3. *Identificar un conjunto de criterios para la comparación de las metodologías.*

Es un objetivo importante del proyecto, que nos permitirá realizar la comparativa de las metodologías ágiles. Hay que determinar unos parámetros, criterios y métricas que nos permitan realizar la comparativa desde un prisma científico y objetivo y teniendo en cuenta las tecnologías concretas en las cuales desarrollamos el proyecto.

4. *Realizar comparativa metodologías ágiles seleccionadas.*

Este es uno de los objetivos principales del proyecto y tiene el propósito de mostrar las diferencias entre las metodologías ágiles seleccionadas y ver las diversidad que actualmente nos ofrece el panorama del desarrollo del software actual.

5. *Caso práctico de aplicación.*

Este objetivo pretende que experimentemos en una situación práctica, los métodos, técnicas y herramientas que podemos encontrar en el desarrollo de un proyecto con Java EE y metodologías ágiles.

6. *Identificación de herramientas necesarias en el desarrollo del software*

El último objetivo que presentamos en el proyecto, consiste en la identificación de las diferentes herramientas que nos podemos encontrar en el desarrollo de una aplicación software y en concreto con metodologías ágiles y/o Java EE. Una tarea que debemos realizar para la consecución de este objetivo es la de encontrar una clasificación que nos permita detectar los principales grupos de herramientas, ya que enumerar todas las herramientas existentes es una tarea prácticamente imposible de realizar.

Estructura del documento

Este documento está dividido en 11 capítulos, entre los cuales podemos encontrar el indice, la introducción, la conclusión y los anexos. Excluyendo estos capítulos que hemos mencionado, el resto se corresponden con objetivos concretos del proyecto o que ayudan a la consecución de los mismos.
Cada capítulo presenta la misma estructura, empieza con una breve introducción que presenta los contenidos que podemos encontrar en el capítulo y también incluye un índice detallado de las diferentes secciones que contiene. Al final de cada capítulo podemos encontrar una breve reflexión o conclusión, donde se pretende realizar una síntesis del mismo y presentar valoraciones personales del mismo. Acto seguido podemos encontrar las referencias bibliográficas a las cuales hacemos referencia a lo largo de cada capítulo.
La numeración de tablas, figuras, notas de pie de página y referencias bibliográficas, se reinicia en cada sección con la intención de facilitar su seguimiento, ya que podríamos encontrarnos en últimos capítulos con números excesivamente elevados y que dificultarían la lectura del documento.
Al final del documento se han incluido unos anexos que son referenciados a lo largo del documento, están formados por diferentes documentos que hemos considerado de interés, algunos hechos personalmente y otros que provienen de fuentes externas.
La elección del color verde para los títulos de las diferentes secciones, así como para la mayoría de diagramas, tablas y figuras generadas, es debido al color representado en el logo de la empresa en la cual se ha realizado el documento, rogamos disculpen si las combinaciones no son todo los estéticas que un servidor habría deseado.

3. Proyectos Java EE

En este capítulo hablaremos de los proyectos software, la plataforma Java EE y como conseguir aunar estos dos conceptos de forma exitosa dando lugar a aplicaciones empresariales. También buscaremos caracterizar las diferentes soluciones empresariales que podemos desarrollar con Java EE, indicando sus principales características.

Contenidos de la sección

Qué es un proyecto

Juan Palacio describe de una forma muy interesante el término proyecto, en su libro *Flexibilidad con Scrum* [1] y lo presenta de la siguiente manera, existen dos conceptos similares y que ayudan a entender mejor el significado de proyecto, estos son los productos y los servicios. Para cada uno de estos conceptos, nos podemos encontrar dos situaciones diferentes:

1. Que los productos sean desarrollados "a medida", partiendo de un diseño y la posterior ejecución de un plan de ejecución y del mismo modo, que los servicios puedan ser actuaciones únicas y específicas, concebidas y realizadas para las necesidades de la ocasión.
2. Que los productos sean el resultado en serie de cadenas o procesos de producción y los servicios, procedimientos normalizados, ejecutados según protocolos y prácticas estandarizadas, que con carácter repetitivo se emplean siempre para prestar el mismo servicio, o servicios del mismo tipo.

Cuando nos encontremos en cualquiera de las dos situaciones anteriores y el resultado obtenido sea un producto, a este proceso lo llamaremos proyecto, pero cuando el resultado sea un servicio, entonces lo llamaremos operaciones. Es importante ver que pese a que comparten características comunes, una operación se ejecuta de forma repetitiva y genera un resultado con características similares, mientras que un proyecto, siempre genera un resultado único.

Los proyectos tienen las siguientes características:

- Los realizan personas.
- Se ejecutan con recursos limitados.
- Se llevan a cabo siguiendo una estrategia de actuación.
- Producen un resultado único.
- Se desarrollan en un marco temporal preestablecido. Tienen un inicio y fin definidos.

Por lo tanto un proyecto es un conjunto de actividades necesarias para producir un resultado previamente definido, en un rango de fechas determinado y con una asignación específica de recursos.

Ejemplos de productos realizados a través de operaciones:

- Un mueble.
- Un ordenador.
- Una prenda de vestir.

Ejemplos de proyectos:

- Construcción de un edificio.
- Desarrollo de un proyecto software.
- Diseño de un nuevo ordenador portátil.

Proyectos Software

Un proyecto software es aquel proyecto que genera como producto final una aplicación o solución software. En la realización de producto se realizan un conjunto de procesos, que conocemos como el ciclo de vida del desarrollo del software.

A menudo se identifican las siguientes nueve fases dentro del ciclo de vida del desarrollo del software:

1. Inicio del proyecto.
2. Especificación de los requisitos.
3. Diseño.
4. Codificación o implementación.
5. Test unitarios.
6. Test de integración.
7. Test del sistema.
8. Test de aceptación.
9. Sistema en uso (Mantenimiento).

Consideramos como un proceso independiente y a la vez vinculado a los otros, el la de gestión del proyecto. Durante el desarrollo del producto se suelen seguir unas directrices o metodologías que guían a los trabajadores a alcanzar los objetivos marcados de la mejor manera posible. A continuación destacamos las prácticas de desarrollo más comunes:

- Metodologías tradicionales:
  Dan nombre a las primeras metodologías que se utilizaron para desarrollar software y que actualmente aun gozan de gran popularidad. A pesar de que el modelo en cascada fue a lo largo del siglo pasado la principal representante de este grupo de metodologías, actualmente podemos considerar el modelo de proceso unificado como su principal valedora. Se caracterizan por realizar un tratamiento predictivo de los proyectos y medir el progreso en términos de artefactos entregados, así como de la especificación de los requisitos, los documentos de diseño, planes de pruebas y revisiones de código.
- Metodologías ágiles:
  Dan una mayor importancia a las personas en vez de a los procesos y se caracterizan principalmente por el uso de técnicas para agilizar el desarrollo del software, así como de una mayor flexibilidad para adaptarse a los cambios en los requisitos del proyecto.

Además de estas dos grandes corrientes metodológicas, podemos encontrar otras prácticas como :

- Cowboy Coding:
  Así se denomna a la ausencia de un método definido; el equipo de desarrolladores hace lo que creen que es mejor en cada momento. También es conocida como método hacker, dado su anárquico modo de proceder.
- ITIL (Information Technology Infrastructure Library):
  Es un conjunto de buenas prácticas para los procesos y operaciones de IT. Los desarrolladores pueden escoger algunos de sus principios sin necesidad de seguirlos todos.
- CMMI (Capability Maturity Model Integration):
  Es una aproximación a la mejora de los procesos usados para guiar el desarrollo de los proyectos. Requiere preparación previa, a menudo la ayuda de un consultor para determinar los procesos y de un asesor externo para evaluar un grupo o proyecto y ver en que nivel se encuentra (valorando del 1 al 5). Originalmente CMMI era estrictamente para la ingeniera del software y la gestión de los proyectos, en su última versión, CMMI, aspira a adaptar aspectos como la implementación de paquetes y infraestructuras. Coincide con ITIL en muchas cosas.

Factores de riesgo en el desarrollo del software

Consideramos factores de riesgo del desarrollo de productos software, a todos aquellos factores que puedan provocar el fracaso del proyecto y consecuentemente la no obtención del producto deseado. Esto incluye la obtención de un producto que no cumple los requisitos del cliente, que no ha sido terminado en los plazos establecidos o simplemente que no es un producto de calidad, con muchos bugs y defectos que lo etiquetan como incompleto. No es nuestro objetivo realizar un estudio exhaustivo de todos y cada uno de los posibles riesgos que nos podemos encontrar en el desarrollo del software, sino enumerar un conjunto lo suficientemente significativo como para que nos sirva a la hora de valorar las diferentes metodologías de desarrollo.
Podemos encontrar diferentes métodos para cuantificar el grado de riesgo de un proyecto [3][5] y multitud de factores que pueden ocasionar riesgos reales en nuestros proyectos [2][6], pero generalmente hay un conjunto de factores de riesgo que deben tenerse en cuenta en cualquier proyecto de desarrollo de software. Me ha gustado especialmente el listado que publicaron James Jiang y Gary Klein [4], indicando el factor de riesgo y quien fue el primer autor en documentarlo.:

Summary of the project development risk

Risk factors	First authors (year)
Project size	McFarlan (1981); Beath (1983); Neumann (1994); Zmud (1980); Barki et al. (1993)
Team size	Alter (1979); McFarlan (1981); Casher (1984)
Technical complexity	Beath (1983); Davis (1982); Zmud (1980); Boehm (1989); Barki et al. (1993)
Technical newness	McFarlan (1981); Beath (1983); Barki et al. (1993)
Application newness	Zmud (1980); Beath (1983)
Team diversity	Alter (1979); Zmud (1980); Casher (1984)
Team expertise	McFarlan (1981); Davis (1982); Boehm (1989)
(Project development)	Barki et al. (1993)
Team expertise	Alter (1979); Anderson et al. (1979); Davis (1982);
(Application)	Boehm (1989); Jiang et al. (1996)
Team expertise (Task)	Anderson et al. (1979); Neumann (1994); Boehm (1989)
Team expertise	Anderson et al. (1979); Boehm (1989); Davis (1982);
(General)	Jiang et al. (1996); Barki et al. (1993)
Experience of leader	Anderson et al. (1979); Boehm (1989)
Number of users	Alter (1979); Alter and Ginzberg (1978); Davis (1982); Barki et al. (1993)
Diversity of users	Alter (1979); Davis (1982); McFarlan (1981)
User attitudes	Alter (1979); Beath (1983); Anderson et al. (1979); Jiang et al. (1996)
Top management support	Anderson et al. (1979); Barki et al. (1993); Jiang et al. (1996)
Resource insufficiency	Boehm (1989); Alter (1979); Anderson et al. (1979)
Task characteristics	McFarlan (1981); Alter (1979)
Conflicts	Anderson et al. (1979); Casher (1984); Barki et al. (1993)

fig 1

A partir del listado anterior y de las fuentes referenciadas anteriormente hemos elaborado el siguiente listado:

- Factor Humano.
  En toda tarea que intervienen las personas hemos de considerar los fallos humanos y todo lo que esto acarrea. En proyectos software tenemos en cuenta sobretodo :
  - Conocimientos, experiencia y habilidades del equipo de desarrollo y del gestor del proyecto.
  - Estado anímico, físico y psíquico del equipo.
  - Diversidad del equipo.

- Factor complejidad.
  Otro factor muy importante que debemos valorar y tener muy en cuenta siempre que hablemos de riesgos asociados a proyectos software, es la complejidad del mismo proyecto, esto nos indicará cuan alerta a los detalles deberemos estar. Un proyecto más complejo

implica una mayor probabilidad de que algo salga mal y por tanto, que el proyecto fracase. Los principales indicadores son:

- Tamaño del equipo de desarrollo.
- Tamaño del proyecto.
- Dependencias del proyecto respecto a equipos externos, internos.
- Características de las tareas.

* Factores tecnológicos.
Debido a que nos encontramos desarrollando proyectos tecnológicos, este factor juega un papel determinante a la hora conseguir nuestros objetivos. Nos pueden afectar de diferentes maneras:

  - Infraestructura insuficiente o inexistente. Ya sea a la hora de implantar la solución o de desarrollarla.
  - Conocimientos técnicos del equipo de desarrollo. De las tecnologías del proyecto.
  - Mala especificación de los requisitos no funcionales. Pueden provocar una situación similar a la primera determinada.

* Factores externos.
Externos al equipo de desarrollo, pero no al proyecto. Incluyen riesgos como:

  - Cliente poco razonable. Puede provocar inestabilidad de los requisitos.
  - En la utilización de COTS[1], el vendedor no se ha responsable de los fallos o bugs de su software.

* Factores gestión.
Son todos aquellos que implican cualquier tipo de intento por controlar los recursos humanos, temporales y técnicos del proyecto:

  - Gestión de los cambios de los requisitos.
  - Planificación basada en el conocimiento. En el conocimiento o experiencia en proyectos similares anteriores.
  - Planificación excesivamente optimista cuando el equipo desconoce la tecnología.
  - Problemas en proyectos anteriores ocupan tiempo al equipo de desarrollo actual.
  - Asignación de las responsabilidades técnicas.
  - Implicación/apoyo del equipo directivo.

1 Commercial, off-the-shelf (COTS) es un término para el software o el hardware, productos tecnológicos en general, que están ya listos para su comercialización. Habitualmente son utilizados como alternativa a productos que podrían realizarse internamente (in-house), cualquier tipo de software empaquetado, se considera un COTS, por ejemplo el paquete ofimático Office.

La plataforma Java EE

Una plataforma Java provee un entorno peculiar en el cual podemos encontrar aplicaciones de cualquier tipo programadas en lenguaje Java. A día de hoy podemos identificar tres plataformas Java:

1. Plataforma Java, Edición Estándar (Java SE).
2. Plataforma Java, Edición Empresarial (Java EE).
3. Plataforma Java, Edición Micro (Java ME).

Cada una de las plataformas Java, contiene una máquina virtual de Java o Virtual Machine(VM) y una interfaz para la programación de aplicaciones (API), esto permite que las aplicaciones escritas para la plataforma, puedan ser ejecutadas en cualquier sistema compatible, con todas las ventajas del lenguaje de programación Java: independencia de la plataforma, estabilidad, potencia, facilidad de desarrollo y seguridad. Generalmente, cuando la gente piensa en Java como lenguaje de programación, esta refiriéndose a la API de Java SE. La API de la plataforma Java SE define las funcionalidades fundamentales del lenguaje de programación Java (tipos básicos, objetos y clases de alto nivel necesarias para la interconexión en red, seguridad, acceso a base de datos, desarrollo de interfaces gráficas y parseo de XML) y esta formada por una máquina virtual , herramientas de desarrollo, despliegue de tecnologías y otras librerías de clases y toolkits usados para el desarrollo de aplicaciones Java. La plataforma Java EE ha sido construida encima de la plataforma Java SE y esta formada igualmente por una API y un entorno de ejecución, pero a diferencia de Java SE, esta enfocada al desarrollo y ejecución de aplicaciones de gran envergadura, multicapa, escalable, fiable y en una red segura. Este conjunto de características definen lo que conocemos como Aplicaciones empresariales, por lo tanto, la plataforma Java EE esta especialmente diseñada para desarrollar aplicaciones empresariales.

Veamos a continuación una breve descripción de las características principales que nos brinda la plataforma Java EE, extraídas de *Your fisrt cup: An Introduction to the Java EE Platform* [7]

Características generales de la plataforma Java EE

La plataforma Java EE representa un estándar para la implementación y despliegue de aplicaciones empresariales. Fue diseñada a través de un proceso abierto y se entablaron conversaciones con un amplio abanico de empresas, lo que permitió que se cubriesen la gran mayoría de los requisitos de las aplicaciones empresariales. Las principales características son:

1. Modelo Multicapa

La plataforma Java EE provee un modelo multicapa y distribuido [fig 2], esto quiere decir que diferentes partes de una aplicación pueden estar corriendo en diferentes dispositivos. La arquitectura define una capa cliente, una capa intermedia (compuestas por una o mas subcapas) y una capa de datos, también conocida como la capa de información empresarial (IE).

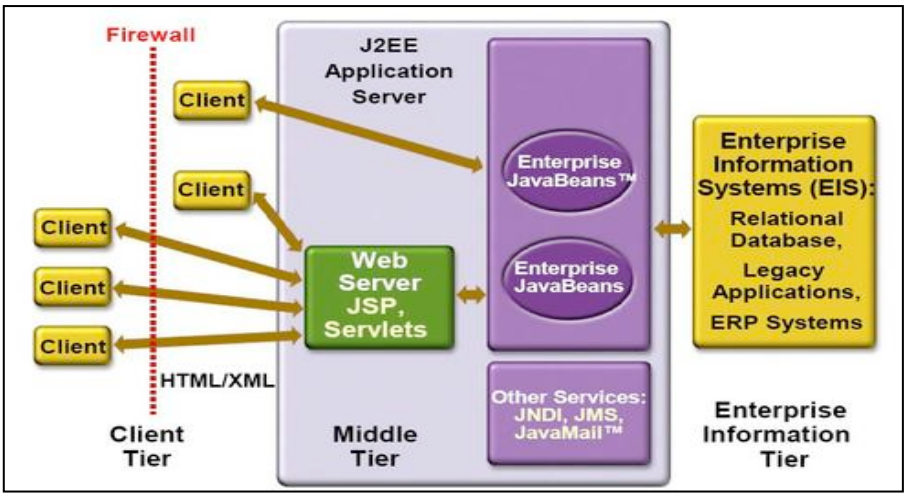


fig 2

Capa cliente

La capa cliente esta integrada por aplicaciones clientes que acceden al servidor Java EE y normalmente se encuentran en una máquina diferente a la del servidor. Los clientes hacen peticiones al servidor y el este procesa las peticiones y las responde. Hay muchos tipos de aplicaciones diferentes que pueden ser clientes Java EE, y no tienen porque ser aplicaciones Java EE, es más, a menudo no lo suelen ser. Los clientes pueden ser un navegador web, una aplicación autónoma[2], o incluso otros servidores que estén corriendo en otra máquina diferente de donde se encuentra el servidor Java EE.

Capa intermedia

El desarrollo de las aplicaciones Java EE se centran en esta capa, con tal de realizar aplicaciones empresariales que sean fáciles de gestionar, más robustas y más seguras. Esta capa puede estar formada por mas de un subcapa, normalmente una capa Web y otra que se centra en los procesos de negocio [fig 3].

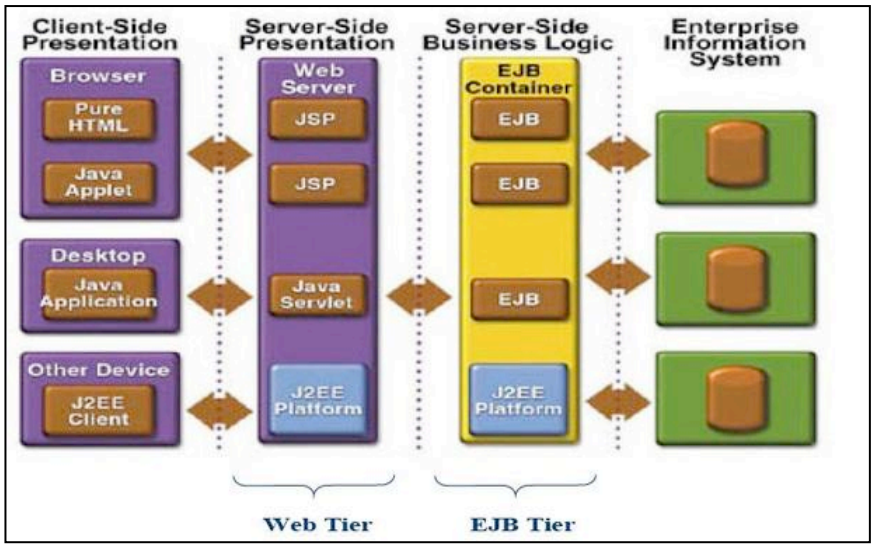


fig 3[3]

2 Utilizamos los términos stand-alone, standalone, autónoma e independiente para referirnos a las aplicaciones que se ejecutan en un equipo local y que no necesitan de otros recursos externos, como por ejemplo la red.

3 Diagramas extraídos del curso on-line disponible en http://www.javapassion.com/j2ee/ e impartido por Sang Shin. [8]

Capa Web

La capa o nivel Web, consiste en el conjunto de componentes que capturan la interacción entre los clientes y la capa de negocio. Sus principales tareas son loas siguientes:

- Generación dinámica del contenido, en varios formatos, para el cliente.
- Recoger las entradas de los usuarios de la interfaz de las aplicaciones cliente y devolver los resultados apropiados desde los componentes de la capa de negocio.
- Controlar el flujo de pantallas o páginas en el cliente.
- Mantener el estado de los datos para una sesión de un usuario.
- Realizar algunas operaciones básicas pertenecientes a la lógica de la aplicación y guardar temporalmente alguna información (usando JavaBeans).

Las tecnologías más comunes de la capa de presentación o Web son:

- Servlets, procesan dinámicamente las peticiones y construyen las respuestas de los clientes, comúnmente utilizado con páginas html.
- JavaServerPages, definen como el contenido dinámico puede ser añadido a las páginas estáticas.
- JavaServer Faces technology, componente framework de la interfaz de usuario que para aplicaciones web, que permite la inclusión de componentes de interfaz de usuario (como pueden ser botones) en una página, convierte y valida datos de los componentes IU, ...
- JavaServerPages Standar Tag Library, una librería de tags que encapsula las funcionalidades principales más comunes de las páginas JSP.
- JavaBeans Components, objetos que temporalmente almacenan lo datos de las páginas de una aplicación.

Capa de Negocio

La capa de Negocio esta integrada por componentes que proveen la lógica de negocio para una aplicación. La lógica de negocio es el código que provee las funcionalidades para un particular dominio del negocio, como puede ser finanzas o un sitio de e-commerce. En un diseño correcto de una aplicación empresarial, las principales funcionalidades se encuentran en los componentes de la capa de negocio.
Las tecnologías más comunes de la capa de negocio son:

- Enterprise JavaBeans (EJB).
- JAX-WS, web service endpoints.
- Java Persistence API entities.

Capa de Datos o del Sistema de Información Empresarial

La capa de datos está integrada por los servidores de bases de datos, sistemas ERP y otras fuentes de datos ya existentes, como mainframes. Habitualmente, estos recursos están situados en una máquina independiente de en la que se encuentra el servidor Java EE, y se acceden a ellos a través de la capa de negocio.

Las tecnologías más comunes de la capa de datos son:

1. Java Database Connectivity API (JDBC).
2. Java Persistence API.
3. J2EE Connector Architecture.

4. Java Transaction API (JTA).

2. *Servidor de aplicaciones Java EE*

Una característica fundamental de la plataforma Java EE es la existencia y la necesidad de utilizar servidores Java EE Este es un servidor de aplicaciones que implementa las APIs de la plataforma Java EE y provee de los servicios estándar de Java EE. Los servidores Java EE son llamados, a veces, servidores de aplicaciones, porque permiten obtener datos de las aplicaciones cliente, del mismo modo que un servidor web sirve páginas web a un navegador. Un servidor Java EE contiene diferentes tipos de componentes, los cuales se corresponden con cada una de las capas de una aplicación multicapa.

3. *Gestión de los componentes mediante contenedores*

Un contenedor de Java EE es una interfaz entre el componentes y las funcionalidades de más bajo nivel que provee la plataforma Java EE para soportar ese componente. La funcionalidad del contenedor esta definida por la plataforma Java EE, y es diferente para cada tipo de componente. No obstante, los servidores Java EE permiten que los diferentes tipos de componentes trabajen juntos para proveer las funcionalidades de una aplicación empresarial. Distinguimos tres tipos de contenedores:

1. Los contenedores Web. Es la interfaz entre los componentes web y el servidor web. Un componente web puede ser un servlet, una página JSP o una página JavaServer Face. El contenedor gestiona el ciclo de vida de los componentes, envías las peticiones a los componentes de la aplicación, y provee de una interfaz a los datos del contexto, como puede ser la información sobre la petición en curso.
2. Los contenedores de la aplicación cliente. Es la interfaz entre las aplicaciones cliente Java EE, las cuales son aplicaciones Java que usan componentes del servidor Java EE, y el servidor Java EE. El contenedor de la aplicación cliente corre en la máquina cliente, y es la puerta de enlace entre la aplicación cliente y el servidor Java EE.
3. Los contenedores EJB. es la interfaz entre los enterprise beans, los cuales proveen la lógica de negocio en una aplicación Java EE, y el servidor Java EE. El contenedor EJB corre en el servidor Java EE y gestiona la ejecución de los enterprise beans de las aplicaciones.

4. *Soporte para los componentes clientes*

Por regla general, los clientes de Java EE suelen acceder a la capa intermedia usando estándares Web (Http,Html,XML) y la plataforma Java EE puede soportar un cierto número de aplicaciones sencillas, gracias a la capa Web y sus componentes tecnológicos. En el caso de existir interacciones más complejas con el usuario, es necesario proveer de funcionalidades directamente a la capa del cliente. Estas funcionalidades son comúnmente implementadas mediante componentes de JavaBeans, que interactuan con los servicios de la capa intermedia via servlets. Los JavaBeans de la capa del cliente se obtienen a través de un applet y se descargan automáticamente al navegador web. Con tal de evitar problemas con versiones antiguas de la máquina virtual de Java, en los navegadores web, las aplicaciones Java EE se descargan automáticamente y instalan los plug-in de Java. Del mismo modo, los beans de la capa de cliente, también pueden ser contenidos por aplicaciones independientes del navegador, pero que hayan sido escritas en Java. En este caso, el cliente esta disponible para ser descargado utilizando Java Web Start Technology, lo que garantiza que siempre tengas la última versión del cliente instalado en tú equipo.
Otras aplicaciones, que no están implementadas en Java, pueden dar los servicios de la plataforma Java EE a los usuarios, gracias a que los servicios son prestados por servlets en la capa intermedia hasta la capa del cliente usando el protocolo HTTP y de este modo es sencillo acceder a ellos desde prácticamente cualquier programa y sistema operativo.

5. *Soporte para los componentes lógicos de negocio*

La lógica de negocio se suele implementar en la capa intermedia como Enterprise Java Beans (también conocidos como Enterprise beans). Los Enterprise Beans permiten a los componentes o a los desarrolladores de aplicaciones concentrarse en la lógica de negocio, mientras el contenedor EJB se encarga de entregar un servicio fiable y escalable.

6. *Soporte de Java EE estándar*

El estándar de Java EE esta definido a través de un conjunto de especificaciones. Entre las principales encontramos las especificaciones de Java EE, la especificación de los enterprise Java Beans, los Java Servlets y las JavaServer Pages (JSP). Además de las especificaciones, otras tecnologías dan soporte al estándar de Java EE, como pueden ser el test de compatibilidad de Java EE, las referencias de implementación de Java EE y el SDK de Java EE.

7. *Puntos positivos y negativos de la plataforma Java EE*

Los principales beneficios de utilizar la arquitectura Java EE son:

1. Disponemos de un amplio abanico de posibilidades para la selección de servidores, herramientas y componentes.
2. Es fácilmente escalable y adaptable a diferentes situaciones empresariales.
3. Tiene un modelo de seguridad flexible.
4. Facilita la división del trabajo y su desarrollo.
5. Write-Once-Run-Anywhere, escribe una vez y ejecútalo donde quieras. Al basarse en el lenguaje de programación Java y la plataforma de Java Standard Edition, hereda sus virtudes y defectos, entre los que se encuentra la portabilidad.
6. Mantiene un estándar, de referencia, para el desarrollo de aplicaciones empresariales.
7. Facilita la integración con sistemas de información existente.
8. Un "producto" maduro, con gran soporte, ejemplos y soluciones probadas.

Algunas de sus debilidades, quejas y complejidades:

1. Largo proceso de estandarización. Desde que se crea un JSR (java specification request) hasta que la especificación esta lista, pueden llegar a tardar entre 2 y 3 años , y eso sin contar con el tiempo añadido de implementación, que por ejemplo para IBM suele ser de 2 años. Este hecho provoca que ante la aparición de un nuevo problema, las empresas no puedan esperar tanto tiempo y pongan en práctica una solución no estandarizada.
2. Demasiado "añadido", por ejemplo a la hora de utilizar los EJB. Este problema se ha solucionado, parcialmente, con la nueva versión de EJB (EJB 3.0), pero aun hay muchas compañías que no utilizan Java EE 5 y por lo tanto aún heredan este exceso de especificación, que es un factor denominador de Java EE.
3. Utilización de contenedores que implementan características que no forman parte del estándar, provocando quebraderos de cabeza en la migración a otro contenedor.
4. Errores, erratas o una mala redacción de la especificación pueden ocasionar que dos implementaciones distintas cumplan la especificación, con todo lo que esto supone. En general aun queda mucho por mejorar la especificación actual.
5. A pesar de ser un estándar para el desarrollo de soluciones empresariales, no todas las empresas necesitan el grado de escalabilidad que ofrece Java EE y otras soluciones se ajustan mejor a sus necesidades.

Escenarios aplicaciones Java EE

Una aplicación desarrollada bajo la plataforma Java EE puede estar configurada de múltiples maneras, para nosotros, un escenario de una aplicación Java EE no será mas que una solución concreta, con unas determinadas tecnologías representativas y estándares de la plataforma. Obviamente no reflejaremos todos los posibles escenarios, pero creemos que los siguientes siete escenarios representan bastante bien todas las posibles variantes que nos podemos encontrar:

1. Escenario 1, html client + Jsp/Servlets + EJB + JDBC/Connector
2. Escenario 2, stand-alone client + Web Server + EJB + JDBC/Connector
3. Escenario 3, html client + JSP/Servlets + JDBC/Connectors
4. Escenario 4, stand-alone client + Web Server + JDBC/Connectors
5. Escenario 5, aplicación EJB stand-alone + EJB + JDBC/Connector
6. Escenario 6, stand-alone client + JDBC/Connector
7. Escenario 7, solución B2B.

Los escenarios 1 y 2, se caracterizan por presentar una arquitectura de cuatro capas, en las cuales están presentes las que hemos introducido en el apartado anterior (cliente, web, negocio y datos). Los escenarios 3, 4 y 5 están compuestos por tres capas, siendo las capas de cliente, web y datos para los escenarios 3 y 4 y cliente, negocio y datos para la 5. El escenario 6 es el más sencillo ya que tan solo esta compuesto por las capas de cliente y de datos y el escenario 7 es el más característico al comunicar dos plataformas Java EE a través del intercambio de mensajes basaos en XML o JMS.
Si desea obtener más detalles al respecto de estas configuraciones, puede referirse a la sección *anatomías de los proyectos Java EE*, en la cual se especifican las características fundamentales de las anatomías de las cuales son representación cada uno de los escenarios especificados. Una anatomía es la solución abstracta, sin especificación de las tecnologías, tan solo indicando según los requisitos funcionales de la aplicación, la arquitectura más adecuada (1,2,3 o 4 capas y que tipos de capas).

Anatomías de los Proyectos Java EE

En esta sección podremos observar las diferentes anatomías que hemos podido extraer de JavaPassion [8] y que nos muestran las arquitecturas más comunes de los proyectos Java EE. De la misma manera, aprovecharemos para relacionar los escenarios propuestos en la sección anterior y explicarlos con mayor detalle. Veamos entonces las anatomías que caracterizaremos y sus correspondientes escenarios :

- Anatomía de clase 1: Aplicaciones Java EE formadas por 4 capas. (multicapa)
  - Escenario 1, html client + Jsp/Servlets + EJB + JDBC/Connector
  - Escenario 2, stand-alone client + Web Server + EJB + JDBC/Connector

- Anatomía de clase 2: Aplicaciones Java EE formadas por 3 capas. (autónomas o stand-alone y basadas en el Web)
  - Escenario 3, html client + JSP/Servlets + JDBC/Connectors
  - Escenario 4, stand-alone client + Web Server + JDBC/Connectors
  - Escenario 5, aplicación EJB stand-alone + EJB + JDBC/Connector

- Anatomía de clase 3: Aplicaciones Java EE formadas por 2 capas.(stand-alone)
  - Escenario 6, stand-alone client + JDBC/Connector

- Anatomía de clase 4: Aplicaciones empresariales Business-to-Business.
  - Escenario 7, solución B2B.

Cada aplicación empresarial tienes unos requisitos específicos, que hacen más adecuado una anatomía u otra. Algunos de estos requisitos son:

- La necesidad de hacer cambios en el "look" de la aplicación, rápidamente y frecuentemente.
- Obtener una aplicación modular, dividida en diferentes capas que diferencien claramente la presentación de la lógica de negocio.
- La necesidad de disponer de programadores especializados en aplicaciones de back-office y diseñadores para desarrollar la interfaz de usuario.
- La necesidad de ensamblar aplicaciones de back-office usando componentes con diferentes orígenes e incluso componentes off-the-shelf.
- La habilidad de desplegar componentes transaccionales a través de diferentes plataformas hardware y software, independientemente de la tecnología utilizada en la base de datos.

Teniendo en cuenta estos requisitos, describiremos las anatomías anteriormente mencionadas.

1. Anatomía de clase 1: Aplicaciones Java EE formadas por 4 capas.

Cuando desarrollamos aplicaciones empresariales Java EE, con una anatomía de 4 capas, hemos de tener en cuenta que:

1. La aplicación constará de una capa o contenedor web que genera dinámicamente el contenido. Esta parte se presenta en un fichero WAR que empaqueta los ficheros .class, .jsp y .html que definen el comportamiento de la presentación e interacción con los usuarios.

2. Por el lado del cliente se deben de desarrollar todo tipo de aspectos visuales, como presentar los resultados, imágenes, etc. También se debe llevar a cabo la implementación de controles mediante javascript, el uso de xforms y en general cualquier tecnología de cliente (DOM,

AJAX, XForms e incluso rich-user interfaces del lado del cliente), siempre y cuando el cliente sea un navegador Web. Es importante remarcar que la tecnología XML juega un papel muy importante en esta arquitectura, ya que el contenedor Web tiene la habilidad de producir y consumir mensajes XML, habilitando el uso de diferentes tipos de clientes, desde navegadores Web hasta motores dirigidos a mostrar soluciones verticales, por lo tanto el cliente puede ser también una aplicación autónoma o stand-alone.

3. La inteligencia de negocio se realiza en la capa de negocio, comúnmente implementada mediante contenedores EJB. En esta parte de la aplicación se debe liberar un fichero EJB JAR.

4. Finalmente, la última capa es la de datos o EIS (Sistema de Información Empresarial), que puede llegar a subdividirse en muchas subcapas e implementar modelos de persistencia muy complejos (grid distribuido) o muy simples (una única base de datos).

Un ejemplo de escenario formado por 4 capas es el que representa la figura 4. Podemos observar como el contenedor Web aloja los componentes Web, el contenedor de EJB aloja componentes de la aplicación que utilizan los recursos del EIS para servir los datos pedidos desde los componentes de la capa Web.

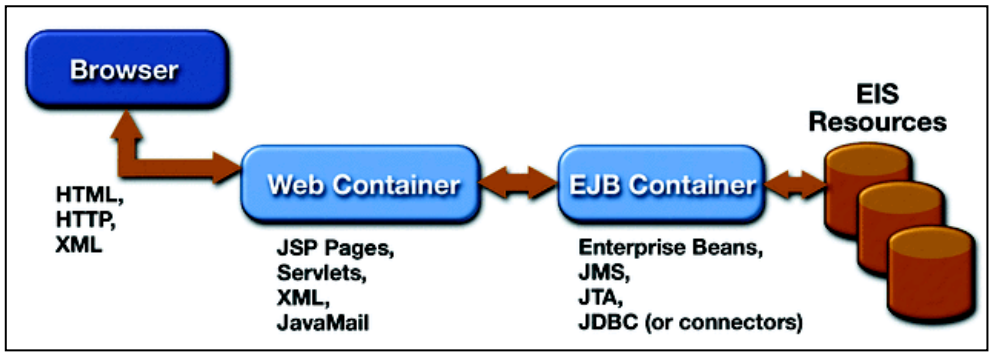


fig 4

Otro escenario ejemplo de una arquitectura de 4 capas es el que mostramos en la figura 5. Un cliente independiente, implementado en Java o en otro lenguaje de programación, consume contenido dinámico Web (normalmente datos en mensajes XML). En este escenario, el contenedor Web, básicamente trabaja con transformaciones XML y provee conectividad Web a los clientes. La lógica de la presentación de la información es responsabilidad de la capa del cliente. La lógica de negocio se implementa mediante enterprise beans, que serán los encargados de acceder a los recursos de la capa de datos o EIS.

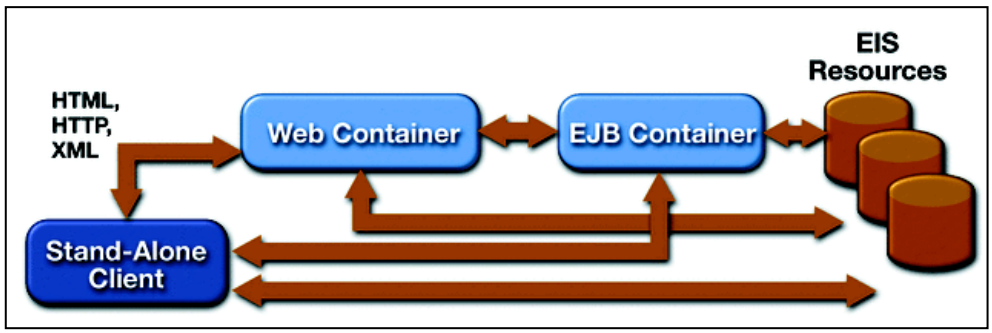


fig 5

Esta arquitectura favorece el desacoplo entre el acceso a datos y la interfaz de usuario de la aplicación.

Es implícitamente escalable y las funcionalidades de back-office están relativamente aisladas de el "look and feel" que presentas al usuario final. Es importante remarcar que la tecnología XML juega un papel muy importante en esta arquitectura, ya que el contenedor Web tiene la habilidad de producir y consumir mensajes XML, habilitando el uso de diferentes tipos de clientes, desde navegadores Web hasta motores dirigidos a mostrar soluciones verticales.

5. Anatomía de clase 2: Aplicaciones Java EE formadas por 3 capas.

Cuando desarrollamos aplicaciones empresariales Java EE, con una anatomía de 3 capas, hemos de tener en cuenta que:

1. La aplicación siempre dispondrá de una capa de datos o EIS.
2. El cliente puede ser tanto un navegador Web como un aplicación stand-alone, EJB o no.
3. En el caso de que el cliente implementado no sea una aplicación EJB stand-alone, se utilizará un servidor web para la gestión de la interacción con la capa de cliente y la implementación de la lógica de negocio.
4. Si el cliente implementado es una aplicación EJB stand-alone, la lógica de negocio será implementada mediante enterprise beans.

Los escenarios derivados de la utilización de una arquitectura de 3 capas y teniendo en cuenta los aspectos comentados son:

- Escenario 3, html client + JSP/Servlets + JDBC/Connectors
- Escenario 4, stand-alone client + Web Server + JDBC/Connectors
- Escenario 5, aplicación EJB stand-alone + EJB + JDBC/Connector

El escenario 1 se corresponde con la figura 6, y es el escenario que comúnmente se reconoce para aplicaciones basadas en el Web. En este caso el contenedor web aloja tanto la lógica de negocio como la de presentación, y se asume que se utiliza JDBC y la arquitectura de conectores de J2EE para acceder a los recursos del EIS.

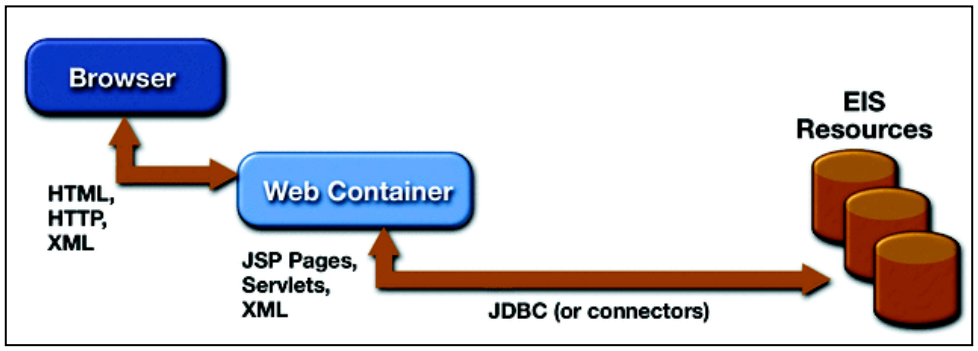


fig 6

Para ver como es el escenario 2, podemos referirnos a la figura 5. En este caso tenemos un cliente stand-alone que se comunica con el contenedor Web, y que aloja la lógica de presentación. El contenedor Web se encargará de la lógica de negocio y de acceder directamente a los recursos del EIS.
En el caso del escenario 3, volvemos a referirnos a la figura 3. El escenario 3 esta formado por clientes EJB que interactuan directamente con los enterprise beans del EJB container, mediante llamadas remotas RMI-IIOP. El servidor de EJB accede directamente a los recursos del EIS utilizando JDBC y conectores de Java EE.

5. *Anatomía de clase 3: Aplicaciones Java EE formadas por 2 capas.*

Las aplicaciones empresariales Java EE, con una anatomía de 2 capas, las conocemos también por ser aplicaciones que siguen la arquitectura cliente-servidor. Podemos observar la figura 4, para detectar las diferentes capas y componentes de esta arquitectura, de la cual solo presentamos un escenario. Este escenario es el que esta formado por una aplicación cliente stand-alone, que accede directamente a los recursos de EIS a través de JDBC o conectores. En este escenario la lógica de negocio como la de presentación, residen en el cliente y están integradas en una sola aplicación.
Esta anatomía es muy utilizada en entornos distribuidos que necesitan un fácil mantenimiento y tienen necesidades de escalabilidad.

6. *Anatomía de clase 4: Aplicaciones Business-to-Business*

En la figura 7 podemos observar un escenario de B2B. Este escenario se centra en la comunicación entre iguales, contenedores Web o de EJB. En el caso de contenedores Web, la comunicación se suele realizar mediante mensajes XML sobre http y se suele utilizar en el desarrollo y despliegue de soluciones de comercio. En cuanto al nivel de comunicación entre contenedores EJB, es mucho más utilizado para entornos de intranets.

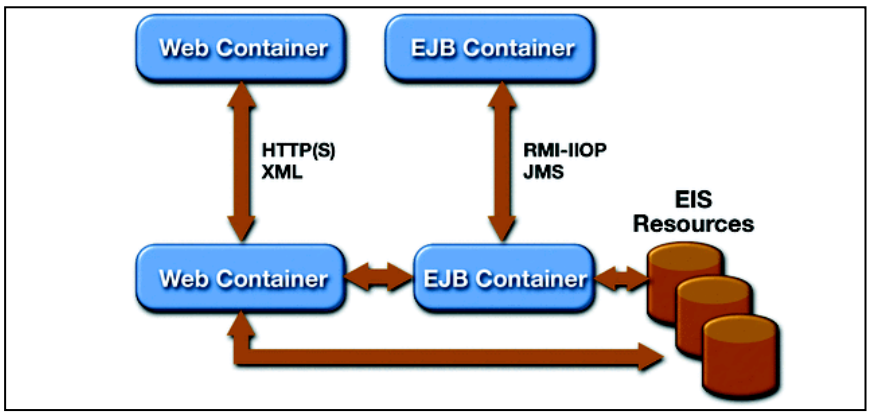


fig 7

Roles típicos en un proyecto Java EE

Tal y como hemos podido observar en el apartado anterior, una de las características de la plataforma Java EE es su modularidad vía componentes, hecho que nos facilita la división del trabajo y su desarrollo. Por tanto, el desarrollo y despliegue de las aplicaciones se puede dividir y repartir en distintos roles, de tal manera que diferentes personas u organizaciones puedan participar en las diferentes fases del proceso. Los dos primeros roles incluyen la adquisición e instalación de productos y herramientas Java EE. Después de que el software es adquirido e instalado, los componentes Java EE pueden ser desarrollados por los proveedores de componentes de aplicación, ensamblados por los ensambladores de aplicaciones, y desplegados por los desplegadores[4] de aplicaciones,. En una gran compañía, cada uno de estos roles puede ser ejecutados individualmente o por equipos. Esta división del trabajo funciona porque cada uno de los roles inmediatamente anteriores, genera un fichero de salida que es utilizado por el rol inmediatamente posterior. Veamos en mayor detalle cada uno de los roles comentados:

- Proveedor de productos Java EE:
  Es la compañía que diseña y pone a la venta las APIs de la plataforma Java EE, y otras funcionalidades definidas en la especificación Java EE. Los proveedores de productos suelen ser vendedores de servidores de aplicaciones que implementan la plataforma Java EE de acuerdo con la especificación de la plataforma Java EE vigente.

- Proveedor de herramientas Java EE:
  Es la compañía o persona que crea herramientas de desarrollo, ensamblaje y empaquetado (packaging) que utilizan los proveedores de componentes para ensamblar y desplegar. Por regla general, los proveedores de productos, también lo son de herramientas.

- Proveedor de componentes de aplicación.
  Es la compañía o persona que crea componentes web, enterprise beans, applets o aplicaciones clientes para usar conjuntamente con aplicaciones Java EE.

  - Enterprise Bean Developer, desarrolla las siguientes tareas para entregar un fichero EJB JAR que contenga uno o mas enterprise beans:

    1. Escribir y compilar el código fuente.
    2. Especificar el descriptor de despliegue
    3. Empaquetar los ficheros .class y el descriptor de despliegue dentro del fichero EJB JAR.

  - Web Component Developer, desarrolla las siguientes tareas para entregar un fichero WAR que contenga uno o mas componentes web:

    1. Escribir y compilar el código fuente de los servlet.
    2. Escribir JSP, JavaServer Faces, y Ficheros HTML.
    3. Especificar el descriptor de despliegue
    4. Empaquetar los ficheros .clas, .jsp, y .html y el descriptor de despliegue dentro del fichero WAR.

  - Application Client Developer, desarrolla las siguientes tareas para entregar un fichero JAR que contiene la aplicación cliente:

    1. Escribir y compilar el código fuente.

4 Traducción literal del término inglés deployer.

2. Especificar el descriptor de despliegue del cliente.
3. Empaquetar los ficheros .class y el descriptor de despliegue dentro un fichero JAR.

- Ensamblador de aplicación:
  Es la compañía o persona que recibe los módulos de la aplicación de los proveedores de componentes y los ensambla en un fichero de aplicación Java EE EAR. El ensamblador o desplegador puede editar el descriptor de despliegue directamente o puede usar herramientas que añaden correctamente los tags XML. Un desarrollador de software lleva a cabo las siguientes tareas para entregar un fichero EAR que contiene la aplicación Java EE:

  - Ensamblar los ficheros EJB JAR y WAR, creados en las fases previas, dentro de un fichero de aplicación Java EE (fichero EAR).
  - Especificar los descriptores de despliegue para aplicación Java EE.
  - Verificar que el contenido del fichero EAR esta bien formado y cumple con la especificación Java EE.

- Encargado de desplegar la aplicación y administrador del sistema:
  Es la compañía o persona que configura y despliega la aplicación Java EE, administra la infraestructura informática donde la aplicación Java EE se va a ejecutar y supervisa el entorno de ejecución. Un desplegador o administrador de sistemas llevan a cabo las siguientes tareas par instalar y configurar la aplicación Java EE:

  - Añadir el fichero de la aplicación Java EE (EAR), creado en las fases precedentes, en el servidor Java EE.
  - Configurar la aplicación Java EE modificando el descriptor de despliegue.
  - Verificar que el contenido del fichero EAR esta bien formado y cumple las especificaciones Java EE.
  - Desplegar (instalar) el fichero de la aplicación Java EE (EAR) en el servidor Java EE.

Aplicaciones empresariales

Las tecnologías de la información en el siglo XXI se encuentran en pleno auge y atraen las miradas y confianza de gran variedad de usuarios, especialmente de las empresas, prometiéndoles una manera de optimizar sus procesos de negocio. El precio que deben pagar las empresas para obtener un alto rendimiento, con ayuda de las tecnologías de la información, es tener un buen sistema de información compuesto por un núcleo organizado de ordenadores, aplicaciones que reflejen las estrategias de negocio, los procesos de negocio, los modelos de negocio y los procesos centrales derivados de las estrategias. Dada la naturaleza intrínseca del mundo empresarial, los procesos y modelos, cambian continuamente, obligando a renovar y actualizar las aplicaciones informáticas, este hecho obliga que el diseño de las mismas aplicaciones comprenda la necesidad de ser rápidamente adaptables y cambiables, cumpliendo los requisitos de negocio de cada momento.
Cualquier organización tiene unos objetivos estratégicos, sobre los cuales se basarán sus decisiones, a través de la persona adecuada. Para hacer esto, esta persona debe disponer de la correcta, apropiada y relevante información en el momento justo. Además de los procesos de negocio principales, se deben de tener en cuenta el soporte que se les da a los clientes, proveedores, monitorear la actividad de los empleados y mejorar las relaciones entre ellos... Considerando todos estos factores, llegamos a la conclusión, de que una aplicación empresarial debe ofrecer una mejora no solo en el rendimiento de la empresa, sino que también se debe adaptar a las características de los procesos de negocio y otras particularidades de la empresa.
Una aplicación empresarial esta especialmente diseñada para resolver los problemas más comunes que se presentan en las grandes empresas, aunque no solo las grandes corporaciones, agencias o gobiernos se pueden beneficiar de sus múltiples beneficios, también las pequeñas empresas y desarrolladores autónomos, que poco a poco ven una creciente tendencia del uso de las aplicaciones distribuidas (través de la red).
Una aplicación empresarial puede estar compuesta por una o más de una aplicaciones corporativas, como puede ser una aplicación que comprende finanzas, contabilidad y recursos humanos o tan solo recursos humanos y frecuentemente proveen funcionalidades de negocio críticas.
Con el objetivo de determinar las principales aplicaciones empresariales que se pueden desarrollar, es importante conocer que es la arquitectura de una empresa y en que consiste, también es importante remarcar que no todas las empresas disponen de una arquitectura de empresa.

La Arquitectura de la empresa

Es el conjunto de elementos organizacionales (objetivos estratégicos, departamentos, procesos, tecnología, personal, etc.) que describen a la empresa y se relacionan entre sí garantizando la alineación desde los niveles más altos (estratégicos) hasta los más bajos (operativos), con el fin de optimizar la generación de productos y servicios que conforman la propuesta de valor entregada a los clientes.(fig 8)

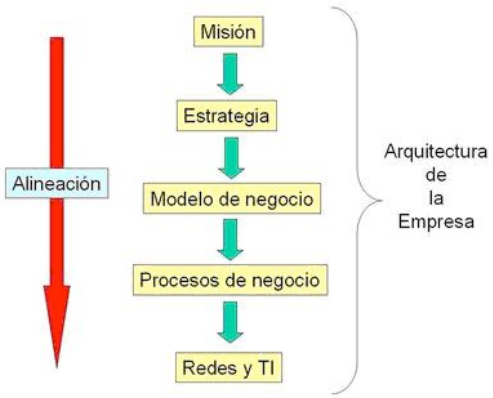


fig 8

Una arquitectura de empresa basada en ordenadores provee muchos más beneficios a largo plazo que costes supone su construcción y desarrollo. Desde el punto de vista del cliente que paga por una aplicación empresarial, inicialmente, puede considerar que esto no es así, pero los beneficios que lleva consigo la incorporación de esta herramienta son palpables:

- Provee un eficiente, apropiado y oportuno soporte para los procesos de negocio.
- Facilita la interoperabilidad entre aplicaciones desarrolladas tanto dentro como fuera de la empresa.
- Facilita la reingeniería de los procesos de negocio.
- Maximiza las posibilidades de explotación de la información y los datos corporativos.
- Asegura la supervivencia de la información/datos.
- Asegura un conveniente soporte a los usuarios, ya sean internos o clientes.

A pesar de que muchas empresas utilizan la arquitectura de la empresa, no todo sus procesos de negocio están incorporados y funcionan de modo independiente, provocando que la comunicación entre los diferentes procesos sea costosa. Una empresa puede disponer de un sistema software para cualquier proceso independiente que concierna a su correspondiente departamento: compras se centra en compras, producción en producción,... Pero para sobrevivir competitivamente es esencial conocer las necesidades de los clientes y ser capaz de responder a ellas, rápidamente. Esto requiere una integración global de todos los proceso de negocio.
Aquellas compañías que son capaces de alcanzar una integración consistente de los procesos de negocio, de una manera flexible, con los socios y clientes, uniformemente a través de toda la compañía, son llamadas empresas en "tiempo-real". Estas empresas pueden responder a las necesidades de los clientes y a los cambios del mercado rápida y eficazmente.

Características comunes aplicaciones empresariales

A continuación listamos las principales características que podemos encontrar en la mayoría de las denominadas aplicaciones empresariales:

- La aplicación se encuentra dividida en diferentes capas, como mínimo una para presentación, otra para la lógica de negocio y otra para la gestión de la información.
- La utilización del navegador Web como cliente de la aplicación es cada vez más común, dando lugar a aplicaciones empresariales basadas en la Web.
- La utilización de componentes como Microsoft +COM Services/ .NET Enterprise Services o Enterprise Java Beans (EJBs)
- Utilización de servidores dedicados como servidores web, servidores de aplicaciones, servidores de integración, servidores de bases de datos y servidores de directorio.
- Se pueden distinguir tres partes en las aplicaciones empresariales actuales:
  - Clientes. Distinguiendo dos tipos de clientes:
    1. Navegador web, usado desde Internet o una intranet.
    2. Cliente local, instalado en cada ordenador.

    Pudiendo coexistir ambos tipos de clientes.

- Infraestructura de la aplicación. Puede incluir Web Servers y diferentes bases de datos

- Extranet o sitios remotos. Muchas empresas disponen de diferentes sucursales geográficamente dispersas (Tokio, Nueva York, Barcelona, …) y deben tener aplicaciones y bases de datos en estas diferentes localizaciones físicas. Estos sitios puede ser que estén conectado vía una Wide Area Network (WAN).

Tipos de soluciones empresariales

El principal punto, a partir del cual determinar las diferentes aplicaciones empresariales que podemos desarrollar, es buscar las necesidades de las empresas. Tal y como hemos visto en los puntos anteriores, los procesos de negocio son una fuente de requisitos y si lo juntamos con la organización de una empresa, podemos obtener prácticamente la totalidad de las principales aplicaciones.
Durante las dos últimas décadas, las compañías han ido gradualmente dejando de desarrollar y mantener internamente sus propias aplicaciones empresariales y sistemas, cambiando su estrategia hacia el uso de sistemas comerciales empresariales, ofrecidos por empresas como SAP, Oracle, Microsoft…, es por esto que una de las mejores opciones es mirar directamente que soluciones empresariales nos ofrecen las principales compañías que trabajan con sistemas ERP[5]. Hay que tener en cuenta que no obtendremos todas las soluciones empresariales que se pueden generar, eso es imposible, por lo que debemos dejar bien claro que existen una gran cantidad de aplicaciones que desconocemos y que necesitarán de un grado de personalización elevado.
De entre todas las posibles compañías escogimos las siguientes:

SAP, Microsoft[6], IBS, y Oracle

Mostramos a continuación las soluciones empresariales seleccionadas y en cuales de las compañías escogidas las hemos identificado.

	SAP	Navision	IBS	ORACLE
Gestión de los recursos humanos. (HCM)	★	★		★
Gestión Financiera.	★	★	★	★
Gestión de la relación con los clientes.(CRM)	★	★	★	★
Gestión de la relación con los proveedores.	★			
Gestión del proceso de distribución.	★	★	★	★

5 Siglas de Enterprise Resource Planning, es el "planning" de como los recursos de negocio (materiales, empleados, clientes,...) son adquiridos y movidos de un estado a otro. Un sistema ERP es una solución empresarial comunmente utilizada en empresas del cualquier tamaño y sector de negocio.

6 El producto de Microsoft para soluciones empresariales de tipo ERP es Navision.

	SAP	Navision	IBS	ORACLE
Gestión de la producción y desarrollo de productos y servicios.	★	★		
Desarrollo del producto y fabricación.	★	★	★	
Aprovisionamiento y tareas logísticas.	★	★	★	★
Business Intelligence.	★	★	★	★
Ventas y servicios.	★	★	★	
Gestión de proyectos.		★		★
Gestión de los activos de la empresa.				★
Servicios de la empresa.	★			★
Espacios colaborativos.		★		
Outsourcing.				★
Aplicaciones de autoservicio.				★

tabla 1

Para poder considerar completo el estudio debemos añadir tres soluciones más, las cuales llamaremos procesos de negocio personalizados:

1. Innovador, nuevo en la empresa y en el mercado.
2. Existente en el mercado, pero no en la empresa.
3. Existente ya en la empresa y se requiere una mejora, añadir funcionalidades ...

Descripción de las soluciones empresariales

A continuación describiremos las diferentes soluciones empresariales Java EE que seleccionamos previamente. Para cada una de las soluciones especificamos sus principales funciones, los escenarios más adecuados (no especificamos todos los escenarios en los que pueden ser utilizados) y algunas observaciones extraídas de sus funcionalidades.

1. *Gestión de los recursos humanos*

- Descripción funcional
  - Procesos de contratación.
  - Procesos de selección.
  - Promociones.
  - Planes estratégicos de contratación.

- Gestión de los programas educativos y de desarrollo profesional.
- Evaluación y gestión del rendimiento del personal.
- Comunicados confidenciales a los empleados.
- Gestión de las relaciones del personal.
- Compensaciones, pensiones, bonus, etc …

* Escenarios compatibles
  - Escogemos el escenario 1, debido que es recomendable la implementación de una aplicación Web. La lógica de negocio es bastante compleja y realizamos acceso a diferentes bases de datos e interaccionamos con diferentes sistemas heredados (legcy systems). También podemos utilizar el escenario 2, si la solución no es una aplicación Web. En ambas soluciones se gestiona de forma independiente la lógica de presentación de la de negocio y el acceso a la base de datos se efectúa mediante EJB.

* Observaciones:
  - Aplicación destinada no solo al personal de recursos humanos, sino a todos los empleados.
  - Aplicación basada en el Web, ya que requiere del uso de un gran número de usuarios y acceso desde distintas ubicaciones.
  - Acceso a la capa de datos, escritura y lectura.
  - Información no "operativa", puede ser actualizada una vez al mes.
  - Información confidencial y muy sensible.
  - Información debe ser fiable.
  - Factores difíciles de cuantificar y clasificar, ejemplo: rendimiento de un empleado.
  - Comúnmente se integrará con sistemas ya existentes (Legacy systems).

2. *Gestión financiera*

* Descripción funcional:
  - Libro de contabilidad, que debe guardar todas las transacciones realizadas.
  - Gestión de las cuentas por pagar y las que están por cobrar.
  - Seguimiento de las leyes estatales, auditorias.
  - Monitorizar el estado de las cuentas
  - Evaluación de niveles de riesgo, configuración alarmas.
  - Gestión de los créditos.
  - Acceso a facturas, estados de cuentas e información de pagos, electrónicamente.

* Escenarios compatibles:
  - Escenario 1 y 2. Si deseamos que sea una aplicación Web, consideramos el escenario 1, sino el 2.
  - En casos que no se cree una aplicación financiera con funcionalidades muy complejas, se pueden utilizar los escenarios 4 y 5. En este caso no implementamos un solución basada en la Web, pero mantenemos la independencia de la gestión de la lógica de presentación y la de negocio.
* Observaciones:
  - Elevado número de transacciones.
  - Elevado grado de concurrencia.
  - Aplicación basada en el Web.
  - Acceso a la capa de datos, escritura y lectura.
  - Distinto tipos de información, la no operativa se corresponde con estadísticas e históricos.

- Muy importante la información de cierres.
- Información confidencial.
- Información debe ser fiable.

3. *Gestión de la relación con los clientes*

- Descripción funcional
  - Funcionalidades asociadas a los diferentes CRM:
    - CRM Analítico
      1. Diseñar y ejecutar campañas de marketing
      2. Diseñar y ejecutar campañas específicas de los clientes, incluyendo ventas cruzadas y fidelización de clientes.
      3. Análisis del comportamiento del cliente con el objetivo de tomar decisiones sobre los productos y los servicios (ej:desarrollo de nuevos productos, precios, …)
      4. Ayuda a la toma de decisiones.
      5. Predicción de la probabilidad de la pérdida de clientes.
    - CRM colaborativo
      1. Da soporte a los clientes (resolución de preguntas, problemas, …)
      2. Mejora el servicio a los clientes a través de la extracción de la información de todos los departamentos.
      3. Coordinar los servicios "multi-canal".
    - CRM operacional
      1. Soporte a procesos de "front-office":
         a. Venta
         b. Marketing
         c. Servicios
      2. Guardar historia de interacciones con usuarios.

- Escenarios compatibles
  - Los escenarios que permiten una gestión separada de las lógicas de negocio y presentación y faciliten el trabajo con los datos. Esta situación nos lleva a recomendar los dos primeros escenarios 1 y 2, el primero si queremos utilizar una interfaz Web y el segundo, con una aplicación stand-alone.

- Observaciones:
  - Uso extensivo de los datos almacenados.
  - Aplicaciones basadas en el Web.
  - Información no "operativa", en su mayoría.
  - Accesos concurrentes a los datos.
  - Uso extensivo de la información almacenada, data mining.

4. *Gestión de la relación con los proveedores*

- Descripción funcional
  - Analizar las estrategias de suministro y abastecimiento:
    - Documentar y calificar proveedores.
    - Elaborar estrategias de proveedores a partir de las diferentes líneas de negocio.

- Seleccionar proveedores, mediante subastas electrónicas.
- Gestión de contratos ofreciendo reuniendo virtuales, edición de documentos y negociación de contratos.

- Capacitar centro de suministro:
  - Permitir que los proveedores procesen los pedidos, emitan facturas y actualicen especificaciones.
  - Crear, sincronizar y mantener contenidos en las bases internas y externas.
- Afianzar relaciones con proveedores:
  - Guardar listas autorizadas de fabricantes según suministro.
  - Descentralización de los procesos de entrega, manteniendo un control central. Los proveedores se "autosirven".
  - Monitorizar de gastos, relaciones con proveedores y rendimiento operacional.
  - Integrarse a la cadena de suministro

    a. Automatizar resolución de contratos.

    b. Generar solicitudes de compra de materiales.

* Escenarios compatibles
  - Escenario 1. Necesitamos tener la lógica de la presentación en una capa independiente, ya que será una fuente de requisitos y de cambios importante, a la vez que debe ser un cliente html para ser fácilmente accesible. Del mismo modo es interesante que la capa de datos sea gestionada por componentes que trabajen bien con datos distribuidos en diferentes bases de datos.

* Observaciones
  - Aplicación basada en el Web.
  - Necesaria una interfaz intuitiva y fácil de utilizar. "Feel and look" elaborado.
  - Acceso a datos distribuidos en diferentes ubicaciones físicas.
  - Los proveedores son usuarios del sistema.

5. Gestión del proceso de distribución

* Descripción funcional
  - Infraestructura de e-marketplace, interconectando a proveedores, partners y clientes.
  - Portal de la cadena de suministro.
  - Pronosticar con precisión la demanda, a partir de la colaboración entre compradores y vendedores.
  - Simular cambios en la cadena se suministros.
  - Monitorizar las etapas de los procesos de la cadena se suministro.
  - Generar alarmas para fallos en los procesos.
  - Generar informes sobre indicadores y objetivos de rendimiento, además de costes y bienes de toda la red de suministro.
  - Integrar procesos de compra en Web, aprovisionamiento basado en reglas, reabastecimiento automatizado y el soporte a múltiples proveedores.
  - Gestionar diversas cadenas de suministro.
  - Verificar disponibilidad de productos.
  - Soportar la gestión de pedidos.
  - Gestión del transporte.

* Escenarios compatibles
  - Escenario 7. Una de las características de las cuales queremos gozar nos dirige directamente a este escenario, esta característica es la de interconexión entre diferentes sistemas (infraestructura e-marketplace).

- Observaciones
  - Aplicación basada en la Web.
  - Comunicación entre diferentes sistemas, Webservices.

6. *Gestión de la producción y desarrollo de productos y servicios*

- Descripción funcional
  - Proveer un entorno de trabajo a lo lardo del ciclo de vida de un producto, para gestionar:
    - Las especificaciones del producto.
    - Las facturas y costes del material.
    - Las fuentes de información y su encaminamiento.
    - La estructura de los proyectos.
    - La documentación técnica.
  - Planifica, gestionar y controlar por completo el proceso de desarrollo del producto.
  - Elaboración y gestión de de planes de proyecto, documentos y estructuras de productos de manera colaborativa a través de la Web.
  - Gestión de la calidad del producto a través de todo el ciclo de vida del producto.
  - Gestión de los activos y equipamiento, haciendo un seguimiento de los costes de los activos y sus agregados.
  - Soporte para el negocio móvil.

- Escenarios compatibles
  - Escenario 1. Por motivos de escalabilidad, accesibilidad y presentación, el escenario 1 es el más recomendable para esta solución empresarial.
  - Escenario 2. Aunque casi siempre es recomendable una aplicación Web, gracias a su gran polivalencia y accesibilidad, en casos que la solución vaya a ser utilizada por pocos usuarios, se puede recomendar una aplicación stand-alone. En este caso, la solución no contará con un gran número de usuarios, por lo que podemos recomendar una solución basada en una aplicación autónoma.
- Observaciones
  - Aplicación móvil, orientada a la Web y dispositivos móviles.
  - Lógica de presentación compleja.
  - Requiere de entornos distribuidos y colaborativos.
  - Numero de usuarios reducido.

7. *Desarrollo del producto y fabricación*

- Descripción funcional
  - Planificar la fabricación y las fases inmediatamente posteriores, necesarias para llevar a cabo la orden registrada.
  - Gestión de los procesos de producción, trabajo de los empleados y otros procesos derivados del trabajo en planta.
  - Documentar, monitorizar e inventariar el ciclo de vida de producción.
  - Monitorizar y documentar que el proceso de fabricación cumple con los estándares de calidad.
  - Planificar el mantenimiento de los activos.
  - Monitorizar los parámetros de medio ambiente, para poder certificar su cumplimiento.

- Escenarios compatibles
  - Escenario 1, permite una implementación de la capa de datos más flexible y al mismo tiempo que la presentación de la aplicación sea independiente de la lógica de negocio.
  - Escenario 2 y 4. Opcionalmente también podríamos implementar uno de estos dos escenarios, en ambas soluciones el cliente stand-alone se encarga de la lógica de presentación, delegando la lógica de negocio a la capa inmediata.

- Observaciones
  - Gran parte del trabajo se realiza sobre la recogida de datos generados por procesos que queremos monitorizar.
  - Aplicación que requiere una interfaz intuitiva que represente los valores de una forma ordenada y clara.

8. *Aprovisionamiento y tareas logísticas*

- Descripción funcional
  - Abastecimiento:
    - Solicitud de compra.
      a. Creación de carritos de la compra
      b. Aprobación de un pedido o carrito de la compra.
    - Procesado de pedidos.
      a. Convertir necesidades en pedidos, automática o manual.
    - Crear, gestionar y buscar en catálogos de productos de proveedores.
    - Gestión contratos de envío.
    - Colaboración con proveedores.
      a. En la creación de pedidos.
      b. Confirmación de recepción de bienes y realización de servicios.
      c. Procesar facturas.
  - Inventario y gestión del almacén:
    - Cross-docking, tanto planeado como oportunista.
  - Logística (entradas, salidas y transporte):
    - Procesado de los movimientos internos y del almacenaje de nuevos productos en el almacén.
    - Gestión del inventario. Visibilidad del stock, …
    - Gestión de estrategias de distribución de material en almacén, FIFO, LIFO y otras más complejas.
- Escenarios compatibles
  - Escenario 7 y 1. Por una parte utilizamos el escenario 7 para colaborar con sistemas externos y por otra el escenario 1 para desarrollar las otras soluciones como inventario y gestión del almacén.
- Observaciones
  - Diferentes bases de datos.
  - Continúo acceso a los datos.
  - Datos operacionales, se debe disponer de datos actualizados al momento para poder tomar las decisiones correctas.

9. *Inteligencia de negocio (Business Intelligence)*

- Descripción funcional
  - Identificar, integrar y analizar datos empresariales dispares procedentes de fuentes heterogéneas.

- Gestión de informes y análisis.
- Personalizar el modo de acceder a la información.

* Escenarios compatibles
  - Al estar enfocado a un grupo muy reducido de usuarios, podemos recomendar una solución stand-alone. Los escenarios 6 y 4, serían dos de los escenarios más adecuados para este tipo de soluciones, sin llegar a utilizar EJB APRA el acceso de los datos.
* Observaciones
  - Principal objetivo es facilitar la toma de decisiones.
  - Mostrar la información del modo más conciso posible y facilitar el acceso al detalle si es requerido.
  - Información no operacional, puede actualizarse en periodos de tiempo más o menos largos.
  - Acceso a diferentes fuentes de información.
  - Dirigido a un número muy reducido de usuarios.
  - Data mining.
  - Se trabaja sobre datos ya filtrados para los motores de búsqueda.

10. Ventas y servicios

* Descripción funcional
  - Seguimiento de las cuentas.
  - Ventas por Internet.
  - Gestión de subastas
  - Crear y procesar órdenes, incluyendo precios y planificación de la realización de las órdenes.
  - Seguimiento de órdenes.
  - Crear y procesar presupuestos.
  - Especificación de precios y tasas.
  - Configuración de los productos.
  - Gestión de los contratos de venta con los clientes.
  - Servicios de post-venta:
    - Gestión de los contratos de servicios.
    - Gestión de la garantía y reclamaciones.
    - Gestión de los servicios de reparación.
  - Servicios de entrega, envío de los productos:
    - Gestión de costes de transporte.
    - Facturación.
    - Planificación de estrategias.

* Escenarios compatibles
  - Escenarios 1 y 2. Es una solución muy compleja, que puede incluye muchos y diversos procesos, es recomendable utilizar un escenario escalable y fácilmente modificable.

* Observaciones
  - Aplicación basada en el Web.
  - Gran número de usuarios.
  - Aplicación compleja y muy completa.
  - Utilización de diversas fuentes de datos.
  - Acceso concurrente a los datos.

11. *Gestión de proyectos*

- Descripción funcional
  - Planificación de proyectos.
  - Estimación de costes asociados a un proyecto.
  - Planificación de actividades.
  - Detallar las tareas de un proyecto.
  - Asociar/Fijar costes a diferentes fases del proyecto, o a la totalidad del proyecto.
  - Seguimiento de tareas específicas o proyectos.
  - Trabajar con diferentes unidades (monetarias, de tiempo, …)
  - Creación de facturas.
  - Diferentes usuarios puedan ver y modificar los detalles del proyecto.

- Escenarios compatibles
  - Escenarios 3, 4 y 5. El escenario 3 en caso de querer crear una aplicación web y el escenario 4 para una aplicación stand-alone. Podríamos utilizar los escenarios 1 o 2, si queremos dar mayor énfasis al diseño de la aplicación.

- Observaciones
  - Interfaz elaborada.
  - Multiusuario.
  - Aplicación ágil.
  - Buena integración con software de CAD o propio de los productos que se ofrecen.

12. *Gestión de los activos de la empresa*

- Descripción funcional
  - Diseño y planificación de las inversiones.
  - Análisis de los activos y mejora de su rendimiento:
    - Agrupar las cuentas relacionadas con los activos.
    - Análisis de daños.
    - Presupuestos de costes de mantenimiento.
    - Realizar predicciones.
    - Realizar pruebas simuladas.

- Escenarios compatibles
  - Escenarios 1 y 2.
- Observaciones
  - Interfaz elaborada y sencilla de utilizar.
  - Acceso a múltiples bases de datos.
  - Objetivo es optimizar la gestión de los activos de la empresa.

13. *Servicios de la empresa*

- Descripción funcional
  - Gestión de viajes:
    - Petición de viajes y aprobación.
    - Contratación de viajes en línea.
    - Gestión de los gastos.

- Gestión del entorno, seguridad y salud de los empleados.
  - Seguimiento de productos peligrosos.
  - Identificar y minimizar riesgos de salud.
  - Política de prevención de riesgos.
- Gestión de la calidad:
  - Planes de inspección.
  - Publicar manual de calidad.
  - Gestión de los documento de un modo seguro.
- Gestión del portfolio.

✱ Escenarios compatibles
- Escenario 1. Muy similar a la solución de recursos humanos, donde normalmente se verán incluídos.

✱ Observaciones
- Interfaz atractiva y fácil de utilizar.
- Publicación de contenidos de un modo ordenado y controlado.
- Interrelación con otras soluciones empresariales.
- Disponibilidad de los recursos ofrecidos via Web.

14. Espacios colaborativos

✱ Descripción funcional
- Portal de empleados:
  - Da acceso a información de negocio.
  - Ordenes de venta
  - Facturas
  - Información de clientes.
  - Comunicación interna.
  - Compartir información con clientes.

✱ Escenarios compatibles
- Escenario 1. Es la única solución que dispone de un cliente html, permite un acceso distribuido a diferentes fuentes de información y gestiona de forma independiente la lógica de negocio y la de presentación.

✱ Observaciones
- Aplicación Web
- Diferenciación por roles de usuario.
- Acceso a diferentes fuentes de información.
- Es un aplicación compleja y previsiblemente añadirá más funcionalidades y con un gran número de usuarios.

15. Aplicaciones de autoservicio

✱ Descripción funcional

En función del negocio para el cual se desarrolla la aplicación, observamos diferentes funcionalidades:
- o Dar a consumidores y empresas cliente, acceso online a sus facturas y cuentas. (B2B y B2C)
  - Servicio seguro 24x7 de acceso a sus facturas.
  - Elección html o pdf para ver las facturas.
- Reportes personalizados de las cuentas, determinando periodos de tiempo y parámetros personalizados.

- Ofrecer la posibilidad de aceptar pagos de los clientes a través de diferentes canales.
- Ofrecer funcionalidades vía Web que normalmente sólo se pueden realizar mediante teléfono o contacto personal:
  - Modificar información de una cuenta de cualquier tipo.
  - Buscar información en una base de conocimiento o información.
  - Enviar correos a través de una interfaz web.
  - Contratar y gestionar servicios (ej: Contratar una conexión a Internet)

* Escenarios compatibles
  - Escenario 1 y 7 , en función de la variante de solución que se implemente.(B2B o B2C).
* Observaciones
  - Acceso distribuido a la información
  - Interfaz compleja que requiere un alto grado de usabilidad.
  - Servicios que se ofrecen a través de la Web.

16. Proceso de negocio personalizado, Innovador

* Descripción funcional
  - Consiste en el desarrollo de una aplicación inexistente en el mercado y que responde a unas nuevas necesidades.
  - Funcionalidades por determinar.
* Escenarios compatibles
  - Depende de la solución
* Observaciones
  - Proyectos de alto riesgo.
  - Muy importante la detección de los requisitos.
  - Alto grado de interacción con el cliente.
  - Es un proyecto eminentemente no predecible.
  - Se pueden utilizar analogías con soluciones similares.

17. Proceso de negocio personalizado, existente en el mercado pero no en la empresa.

* Descripción funcional
  - Se procede al desarrollo de una aplicación ya existente en el mercado y que responde a unas necesidades que tienen otras empresas del mismo sector.
  - Funcionalidades por determinar.
  - Escenarios compatibles
  - Depende de la solución.
* Observaciones
  - Se pueden adaptar soluciones existentes en el mercado.
  - Es importante la experiencia desarrollando este tipo de soluciones.
  - Recoger requisitos a través de aplicaciones existentes.
  - La interacción con el usuario será frecuente, pero no tan determinante como con proyectos innovadores.

18. Proceso de negocio personalizado, existente en la empresa, mejora.

* Descripción funcional
  - Existe un producto funcional en la empresa y se quiere sustituir, ya sea porque el actual producto no cubre todas las necesidades, es una aplicación heredada y muy antigua que no permite su ampliación, tiene un elevado coste de mantenimiento, …

- Funcionalidades por determinar.

* Escenarios compatibles
  - Depende de la solución.
* Observaciones
  - Se parte de un producto ya existente.
  - Es fundamental conocer las causas de sustitución del producto existente.
  - El desarrollo de la aplicación puede responder a un proceso predecible.
  - Estos proyectos se dividen en dos partes bien diferenciadas:
    - Adaptación del software existente.
    - Desarrollo de las nuevas características.

    Es recomendable realizar la adaptación en el menor tiempo posible e iniciar las mejoras cuanto antes.
  - La interacción con el cliente es casi tan importante como con los proyectos innovadores, depende del conocimiento de la aplicación que se quiere sustituir y el número de mejoras a introducir.

Realizar proyectos empresariales con Java EE y con éxito

Al inicio de este capítulo hemos visto qué es un proyecto software y los riesgos más comunes que debemos tener en cuenta a la hora de llevarlos a cabo. Mas tarde hemos continuado con una descripción de la plataforma Java EE y las soluciones empresariales que nos permite desarrollar, caracterizándolas mediante sus diferentes escenarios y anatomías, e indicando los aspectos que hemos considerado más remarcables y necesarios de cada uno, a tener en cuenta a lo largo de su desarrollo. Parece pues, el momento propicio para intentar determinar un seguido de puntos que nos garanticen el éxito en el desarrollo de nuestra aplicación empresarial con Java EE, pero teniendo siempre en cuenta el marco teórico en el que nos encontramos y que no es el objetivo fundamental del proyecto, sino un punto necesario mas, para poder evaluar las metodologías ágiles en este entorno empresarial y con unas determinadas tecnologías asociadas.

- El primer punto que debemos considerar es vigilar todos y cada uno de los factores de riesgo que comentamos en la sección de *factores de riesgo del desarrollo del software,* de este modo eliminamos una gran parte de los factores que conseguirían que nuestro proyecto no fuese exitoso.
- Cada proyecto es diferente y es realmente importante darse cuenta de este punto, incluso en proyectos muy similares es complicado recrear el entorno y las situaciones que se dieron la primera vez, por lo que no debemos bajar la guardia y relajar alguna fase del proyecto por muy similar que sea a otro realizado anteriormente, debemos ser detallista y al mismo tiempo entender que no podemos controlar todos los detalles de un proceso que puede llegar a ser muy complicado. En resumidas cuentas, asumir que hay imprevistos que no podremos controlar y que pueden provocar desviaciones en el proyecto, nuestro objetivo debe ser reducir estas desviaciones al máximo, detectándolas cuanto antes mejor y mejorando nuestra capacidad de reacción y de adaptabilidad a los cambios.
- Cumplir el test de "*Joel on Software*"[11]. Con total falta de rigor y de la misma manera que advierte Joel en su post, recomiendo cumplir todos y cada uno de los doce puntos recomendados por Joel. Poco a poco, a medida que avancemos en la lectura de este proyecto, podremos observar como muchos de estos puntos no son anárquicos ni mucho menos y que pertenecen a varias metodologías probadas y contrastadas y que por la época en la que las propuso (año 2000) eran muy novedosas. Estoy refiriéndome a técnicas como la utilización de un servidor de integración continua o repositorios de código, cosas tan básicas actualmente en el desarrollo del software actualmente como puede ser respirar para nosotros.
- Seguir una metodología de desarrollo, sin entrar todavía en cual es más o menos adecuada, recomendamos encarecidamente la utilización de una metodología que ayude a los desarrolladores, gestores de proyectos a nadar en una misma dirección y saber en todo momento lo que hay que hacer.

Bien, hasta ahora todo lo que hemos dicho se puede aplicar a cualquier proyecto de desarrollo de software, pero qué pasa en concreto con los proyectos que se desarrollan utilizando las tecnologías Java EE. Muchos consideran que no existen diferencias entre los proyectos en función de la tecnología que desarrollemos y yo, personalmente me encuentro más cercano a estas ideas, pero también considero interesante ver con mayor detalle que parámetros pueden marcar la diferencia en un proyecto Java EE y darnos ese tan ansiado resultado exitoso y satisfactorio. Por tanto los puntos que vienen a continuación son un compendio de las conclusiones que he extraído de la elaboración de las secciones anteriores y de las siguientes referencias que podemos encontrar por la web[9][10], la primera referencia es una presentación que hizo Humprey Sheil y la segunda es un artículo de Rod Johnson.

1. Entender y conocer la plataforma Java EE y sus tecnologías asociadas. Esta premisa que parece tan evidente debería ser el primer punto a tener en cuenta si queremos desarrollar un proyecto Java Ee con éxito.
2. No caer en la sobre-ingeniería y utilizar patrones de diseño. Es muy común sobrecargar componentes con más funciones de las que debería, provocando que finalmente no sepamos

ni lo que hacen. La utilización de patrones de diseño, generalmente, nos ayudará a obtener un mejor diseño de nuestra aplicación y obtendremos beneficios en la escalabilidad, el rendimiento y el mantenimiento de nuestra aplicación.

3. Separar la lógica de presentación de la lógica de negocio. Nos añadirá una mayor extensibilidad, mantenibilidad y aumentará del rendimiento de nuestra aplicación.
4. Duplicar o simular el entorno de producción donde desarrollas. De este modo evitarás errores extraños, que te costarán mucho tiempo y esfuerzos solucionar.
5. Escoger correctamente a tus proveedores y conocerlos. Entendemos que si escoges bien a tu proveedor, evitarás problemas de compatibilidad en tus herramientas y aumentarás la productividad de tu equipo al utilizar una herramientas que funcionan correctamente. Es tan importante conocer las herramientas con las que trabajas como las tecnologías que utilizas, ya que sino podrías caer en el error de "reinventar la rueda" y perder mucho tiempo implementando funcionalidades que tu proveedor ya te esta ofreciendo, tan solo por desconocimiento de estas.
6. Diseñar nuestra aplicación teniendo en cuenta la tecnología y las exigencias de rendimiento y escalabilidad de la aplicación. Si desatendemos esta práctica, podemos caer en sistemas lentos y que sobrecargan las funcionalidades de la parte del servidor.
7. Utilizar una metodología, prácticas y herramientas que te permitan realizar los builds de tu aplicación de forma eficiente y continuada, y que faciliten la realización de las pruebas. Prácticamente estoy diciendo que uses Extreme Programming, pero atención, no estoy indicando nada mas que una metodología que huya de situaciones rígidas e inalterables y que te permita adaptarte rápidamente a los cambios.
8. Huir de la publicidad y basarse en los hechos, hechos técnicos. La selección de tus herramientas, prácticas y modelos de desarrollo deben estar sustentadas en un estudio serio y contrastado, no debido a modas pasajeras y/o campañas de marketing muy bien orquestadas que convencerían a cualquiera.
9. Evitar a toda costa la baja productividad. ¿Cómo? Pues agilizando los ciclos entre las fases de despliegue y las de pruebas, agilizar el proceso de montaje (builds), disponer del hardware adecuado,... Los puntos 7 y 5 también nos ayudan a incrementar sustancialmente la productividad de nuestro equipo.
10. Frameworks, sí gracias. Es fundamental el uso de buenos frameworks que permitan su personalización. Mires por donde mires, todos te dirán que, por ejemplo, JDBC es bueno y funciona bien, pero es imperativo utilizar un framework para el acceso a los datos. Es un punto sobre el que debemos tener cuidado, ya que aunque nos prometen grandes ventajas su mala utilización puede tener consecuencias nefastas en la consecución de nuestros objetivos.
11. Vigila tu nivel de fanatismo. Parece que tenemos una tendencia innata a idolatrarlo todo, ya sean tecnologías, personas u objetos. Por esto, no es extraño encontrar situaciones en las que forzamos a utilizar una tecnología en un proyecto en el cual otra habría sido más adecuada, tan sólo por nuestra predilección por la primera.
12. Ten un arquitecto "Oryzus"[12]. El rol del arquitecto es un rol fundamental en todo proyecto empresarial con Java EE, debemos evitar encontrarnos con el perfil de arquitecto endiosado o como menciona Martin Fowler en su artículo "Who Needs an Architect"[12], un arquitecto reloadus (de la famosa película de los hermanos wachowski, Matrix Relaoded donde aparecía el arquitecto del sistema Matrix). El arquitecto reloadus es un arquitecto que toma él solo todas las decisiones importantes y lo hace así porque cree que el resto de su equipo no tiene las habilidades como para poder tomar esas decisiones. Estas decisiones se deben de tomar en una etapa temprana del proyecto y se elabora un plan que deberan seguir el resto de los miembros del equipo. El arquitecto Oryzus esta siempre atento a lo que se hace en el proyecto, sigue una intensa colaboración con los miembros de su equipo, ayudando a implementar partes del código de la aplicación, recogiendo requisitos del cliente, ..., el arquitecto Oryzus es una especie de guía o mentor.

Conclusiones

La plataforma Java edición empresarial surgió como repuesta a unas necesidades de desarrollo específicas y poco a poco se ha ido consolidando como una de las alternativas mejor consideradas a la hora de implementar soluciones empresariales. No podemos negar el esfuerzo que se ha hecho desde la comunidad Java y desde la misma Sun para crear cada vez una plataforma mejor, que fuese más fácil de trabajar con ella y se adaptase mejor a los posibles problemas que querían solucionar. A pesar de todo esto, no podemos obviar que las aplicaciones empresariales son complejas y que Java EE es una plataforma con excesivo plumbing o sobreespecificación, que en muchos casos no hace otra cosa más que entorpecer procesos que serían más sencillos.

En este proyecto se ha intentado dar una visión general de la plataforma Java EE y de las posibles aplicaciones que se pueden desarrollar con ella. Las conclusiones a las que he llegado es que a pesar de destinar grandes esfuerzos a buscar las posibles aplicaciones que se pueden desarrollar con Java EE, este es un cometido imposible, simplemente no podemos más que hacer referencia a las soluciones que actualmente están utilizando un pequeño sector de las empresas y siempre sin ver, ni intentar descubrir lo que esta por venir. Del mismo modo, cuando hemos intentado caracterizar estas soluciones funcionalmente, hemos podido comprobar la enorme cantidad de funcionalidades de las cuales se las dotan y que cada vez se dirigen hacia un entorno de red, accesible y colaborativamente efectivo. Por lo tanto, aunque podamos observar un conjunto de funcionalidades comunes y mas o menos bien especificadas, cada empresa adapta estos procesos a su entorno de negocio y los modifica según sus necesidades. Es decir, tenemos una base de requisitos y soluciones que nos puede servir de punto de partida para capturar más requisitos y adaptar la solución a la necesidad individual del cliente.

Al mismo tiempo que definíamos un conjunto de soluciones prácticamente aplicables a cualquier empresa, en cualquier área de negocio (recursos humanos, gestión financiera, gestión de la relación con los cliente…) establecíamos unos escenarios típicos de la anatomía de Java EE. Al intentar cruzar estas dos características nos encontramos con varios detalles dignos de mencionar:

- Un escenario era prácticamente aplicable a cualquiera de las soluciones empresariales. Factores como la expresa petición del usuario para dotar de mayor conectividad a la aplicación o los diferentes recursos disponibles para desarrollar la aplicación, pueden discrimina enormemente la selección.
- Necesitamos más información. Las posibles funcionalidades que típicamente se implementan en la solución, pueden inclinar la balanza de una solución u otra.
- La experiencia es un grado. Otro factor que realmente ayuda a la selección de los escenarios, es haberte enfrentado con este problema tantas veces que ya hayas probado todas las posibles combinaciones y aún así, cada solución como proyecto software que es única.

Como corolario a esta primera introducción al desarrollo de aplicaciones empresariales con Java EE, me gustaría remarcar el concepto intrínseco y asociado de proyecto software, como un proyecto que genera un producto único y prácticamente irrepetible. Este concepto nos induce a pensar que a pesar de desarrollar aplicaciones con funcionalidades muy similares, es difícil recrear los escenarios y características que se dieron anteriormente, siendo necesario dar una mayor importancia a nuestra capacidad para adaptarnos a los cambios que surgirán y dando gracias de que no siempre debamos partir desde cero.

Referencias

[1] Juan Palacio, Flexibilidad con Scrum, principios de diseño e implantación en campos Scrum. SafeCreative. Edición Octubre 2007

[2] Jesal Bhuta, Sudeep Mallick and S. V. Subrahmanya. A Survey of Enterprise Software Development Risks in a Flat World 2007

[3] Mizuno, O., Adachi, T., Kikuno, T. and Takagi, Y. On prediction of cost and duration for risky software projects based on risk questionnaire 2001

[4] J.Jiang, G.Klein. Software Development Risks to project effectiveness. Journal of Systems and Software, 52:3-10, 2000

[5] O.Mizuno, T.Kikuno, Y.Takagi and k.Sakamoto. Characterization of risky projects based on a project managers' evaluation. pages 387-395. 2000

[6] Paul L Bannerman. Software Project Risks in the Public Sector, ASWEC'07. 2007

[7] Sun Microsystems, Inc. The Java Platform Enterprise Edition 2006

[8] Sang Shin. Introduction to Java EE (J2EE)

[9]Humphrey Sheil. J2ee project dangers http://www.javaworld.com/javaworld/jw-03-2001/jw-0330-ten.html 2001

[10] Rod Johnson - Why J2EE projects Fail http://weblogs.java.net/blog/edburns/archive/2005/03/tssjs_rod_johns.html 2003

[11] Joel Spolsky -The Joel Test: 12 Steps to Better Code - http://www.joelonsoftware.com/articles/fog0000000043.html 2000

[12] Martin Fowler. Who needs an Architect? IEE Computer Society2003

4. Calidad del Software

En este capítulo haremos un breve recorrido por las características y atributos que determinan la calidad del software, identificaremos las métricas más significativas para valorar la calidad y veremos brevemente que propuestas existen para garantizar la calidad del software y en que principios se basan.

Contenidos de la sección

Qué es la calidad del software

Calidad: *"Propiedad o conjunto de propiedades inherentes a algo, que permiten juzgar su valor"*[1]

El término calidad parece depender del entorno en el cual se esta utilizando y es susceptible de ser malinterpretado, por tanto, es importante dejar desde un principio bien asentado este concepto. Descartamos el concepto de calidad asociado al lujo o al nivel social, que solemos interpretar cuando hablamos de productos comercializados por marcas reconocibles y de "alto standing" y nos centramos en las características del producto en si y a su cumplimiento de las funcionalidades esperadas, de un modo correcto.

Cuando asociamos los términos calidad y software, no son pocas las diferentes interpretaciones que podemos encontrar, pero observamos que siempre se tienen en cuenta las definiciones de Crosby (1979) que define la calidad como la conformidad a los requisitos[2] y la definición de Juran y Gryna (1970)que la define como la adecuación al uso[3]. Si valoramos la percepción más común de la calidad del software se suele reconocer como la ausencia de "bugs" o fallos que contenga nuestro producto, que no contradice ni mucho menos las definiciones mencionadas anteriormente, ya que un producto con fallos, difícilmente cubrirá las funcionalidades requeridas. Esta definición basada en los fallos del producto se suele expresar como:

1. Ratio de defectos. Número de defectos por millones de líneas del código fuente, por puntos de función u otras unidades.
2. Fiabilidad. Número de fallos durante *n* horas de funcionamiento, tiempo sin fallos o la probabilidad de no tener ningún fallo durante cierto tiempo.

No podemos obviar el rol que esta jugando en la calidad o la percepción de la calidad de nuestro software, el cliente. Es lo que comúnmente conocemos como satisfacción del cliente. Cada compañía monitoriza la satisfacción de diferentes maneras, por ejemplo IBM [1] lo hace utilizando los siguientes niveles:

- CUPRIMDSO[4]
  - Capacidad, usabilidad, rendimiento, fiabilidad, instalabilidad, mantenibilidad, documentación/información, servicios e impresión general.

y HP:

- FURPS[5]
  - Funcionalidad, usabilidad, fiabilidad, rendimiento y servibilidad.

Otras compañías utilizan dimensiones similares para determinar la satisfacción del cliente. A estos atributos, Juran los llama parámetros de calidad.

Podemos considerar también dos conjuntos de objetivos en la calidad de las soluciones software:

- La calidad del producto software.

---

1 Definición de la Real Academia de la Lengua del término calidad.

2 "Conformance to requeriments"

3 Fitness for use

4 Iniciales en inglés de Capability, Usability, Performance, Reliability, Installability, Maintainability, Documentation/information, Service, Overall.

5 Inciales en ingles de Functionality, Usability, Reliability, Performance, Serviceability.

- La calidad de los procesos.

Por la calidad del producto software entendemos todo lo que hemos explicado anteriormente, es decir, básicamente que cumpla los requisitos establecidos por los clientes, presente un bajo ratio de defectos y una alta fiabilidad y tenga una buen grado de satisfacción de los usuarios. Pero cuando hablamos de la calidad de los procesos, estamos persiguiendo el mismo objetivo, obtener un producto de calidad, sólo que queremos llegar a este objetivo mediante un control y una gestión de todo el proceso de desarrollo. De aquí derivamos que si nuestros procesos son de calidad tenemos un mayor número de probabilidades de que nuestro producto final sea también de calidad. Normalmente esta distinción la podremos observar cuando hablamos de mejora de la calidad del software.
Otra descripción que podemos encontrarnos en *Metrics And Models In Software Quality Engineering* [1], es que la calidad del software esta formada por dos niveles:

- La calidad intrínseca del producto.
- La satisfacción del cliente.

A pesar de estas múltiples aproximaciones hacia la calidad del software, la definición de calidad que hemos citado al inicio de esta sección es un factor común para todas y cada una de ellas, ya que no dejamos de definir propiedades del producto software (que cumpla los requisitos, que no tenga fallos, que se adecue funcionalmente,...) y valorándolas, obtenemos la calidad.

Parámetros de la calidad

Ya hemos introducido en la sección anterior algunos parámetros o características de nuestro software que nos ayudan a determinar a la calidad de nuestro producto, pero creemos interesante ahondar un poco más en estos parámetros y en la teoría de la medición de la calidad. Veamos los principales parámetros que determinan la calidad de un producto software y que podemos ver en mayor detalle en Software Quality [2] :

- Fiabilidad.
- Usabilidad.
- Mantenibilidad.
- Adaptabilidad.

Fiabilidad

> *"Probabilidad de que el software funcione sin errores."* [3]

La definición que acuña la Nasa sobre un software fiable es una de las definiciones más intuitivas que podemos encontrar sobre la fiabilidad del software. y hace referencia a la percepción de la fiabilidad como un parámetro que utiliza la teoría probabilística para sustentarse.
La fiabilidad viene determinada por los siguientes atributos:

- Completitud. Un software puede estar incompleto si no se ha implementado alguna funcionalidad, no se ha diseñado o incluso si no se ha llegado a especificar un requisito.
- Consistencia y precisión. Podemos considerar que la fiabilidad es un indicador de cuanto podemos depender de un producto. De este modo, si un producto es inconsistente, a veces funciona y otras veces tiene un comportamiento anómalo, no podremos depender de este software. Del mismo modo puede que confundamos un sistema inconsistente por uno poco preciso, si los fallos que se dan tienen que ver con algún cálculo numérico. Por ejemplo, este caso se puede dar que si no controlamos las divisiones por cero consideremos el sistema como poco precios, cuando en realidad es inconsistente, de ahí la relación de estos dos atributos.
- Robustez. Se refiere a la capacidad del software a responder a entradas no esperadas o usos que para los cuales no esta destinado, y continuar funcionando. Un buen ejemplo sería el de un programa que espera recibir el numero de cuenta de un usuario y le introducimos su nombre, el programa debe detectar el error, responder correctamente y continuar ejecutándose.
- Simplicidad. La simplicidad de los diseños, en la implementación y en el desarrollo del software en general, es uno de los parámetros a los que más esfuerzo suelen dirigir los desarrolladores y profesionales del sector. Esto es así, ya que estructuras complejas de software son más pródigas a tener errores y bugs, que también son más difíciles de detectar y solventar.
- Trazabilidad. La trazabilidad entre el modelo de requisitos de un programa y el producto final es un medio para asegurarnos la completitud. Como uno de los atributos principales de la fiabilidad, la trazabilidad también se refiere a la suficiencia y exactitud de los casos de prueba, usados para determinar como de bien el programa satisface los requisitos especificados para el mismo.

Usabilidad

La usabilidad es el atributo del software que describe la medida en que completamente y convenientemente se llevan a cabo las funciones establecidas en un entorno determinado.[2]
Me gusta mucho mas el término facilidad de uso, que aunque no es exactamente usabilidad, si que podemos determinar que un producto software fácil de utilizar es usable.
La usabilidad viene determinada por los siguientes atributos:

- Fiabilidad. Es la principal condición para que un producto software sea usable. ya hemos hablado de esta característica ampliamente con anterioridad.
- Exactitud. Los resultados deben ser lo más exactos posibles, sino no será usable.
- Claridad y precisión de la documentación. Un producto por muy intuitivo que sea, siempre necesita de una buena documentación que guíe al usuario en su proceso de aprendizaje en su uso.
- Conformidad al entorno operativo. La aplicación debe de funcionar en el entorno en el cual va a ser utilizada y debemos tener en cuenta las diferencias que podamos encontrar respecto a los entornos de desarrollo y pruebas, en los cuales hemos probado el funcionamiento del producto.
- Completitud. Nada que añadir respecto a lo mencionado en el apartado anterior.
- Eficiencia. La eficiencia nos ahorra el uso innecesario de recursos como memoria, capacidad de almacenamiento, tiempo de respuesta, ... e incrementando el feed-back que tenemos sobre la calidad del producto.
- Habilidad de probar el software. Este atributo viene determinado desde la fase de especificación de los requisitos y se aplica a cada una de las facetas del comportamiento del sistema. Ademas este atributo nos asegura que el software especificado en los requisitos se puede usar. Si encontramos un requisito del producto que no se puede probar, eso quiere decir que la persona que determino ese requisito no lo entendía completamente.

Mantenibilidad

"*El esfuerzo requerido para localizar y arreglar un error en un producto software*"[5]
"*La facilidad con la que el software puede ser mantenido*"[4]

Todo informático que se precie, se ha encontrado alguna vez en la disyuntiva de descifrar el código que hizo hace un año o incluso peor, el código que otra persona ha realizado, con las complicaciones asociadas que esto conlleva. Veamos que atributos nos determinan la mantenibilidad de una solución software:

- Precisión y claridad de la documentación. No son desde luego el principal exponente de la mantenibilidad del software, pero realmente ayuda disponer de una buena documentación. Esta documentación suele contener un compendio de diseños de la aplicación y sus estructura, así como información relevante respecto a su instalación y uso.
- Modularidad. La modularidad ayuda a crear una documentación mas fácil de entender, a la vez que facilita a los desarrolladores la aplicación de los cambios necesarios y un mejor entendimiento de la aplicación.
- Legibilidad. Cuando la documentación no es buena o simplemente inexistente, entonces nos dirigimos directamente al código. La utilización de anotaciones, comentarios, si se hace adecuadamente ayuda al entendimiento del mismo y seguir unas guías de estilo que faciliten la compresión y lectura del código.

- Simplicidad. Un atributo del que ya hemos hablado par ala fiabilidad y que ahora nos permite mantener una aplicación con mayor facilidad. Observamos que los atributos de legibilidad o modularidad, son indicativos de la simplicidad de nuestro software.

Adaptabilidad

Medida que indica la facilidad con la que un programa puede ser alterado para adecuarse a diferentes usuarios y limitaciones del sistema.[4]

Cuando observamos detalladamente este parámetro de calidad, detectamos otros tres parámetros nuevos que no habíamos considerado inicialmente, estos son:

- Capacidad de modificar el software. Indica la capacidad de modificar la manera en que una función es llevada a cabo. Esta compuesta por los mismos atributos que el parámetro mantenibilidad.
- Capacidad de extender sus características. Hacemos referencia a añadir nuevas funcionalidades. Otra vez nos encontramos con que tiene los mismos atributos que el parámetro mantenibilidad, mas uno nuevo. Este nuevo atributo hace referencia a la eficiencia de nuestro software, una vez ya hemos optimizado nuestros recursos de memoria, tiempo y otros recursos, entonces estamos preparados para poder añadir nuevas funcionalidades.
- Portabilidad. Describe la facilidad con la que el software puede ser movido de un entorno operacional a otro, de un ordenador a otro o a un sistema operativo diferente en el mismo equipo.

Métricas de calidad

Ya hemos vista las características que determinan la calidad del software, pero si lo que queremos es poder medirla, necesitamos unos parámetros, medidas o métricas que nos permitan determinar el valor de la calidad buscada.
En este ámbito no estamos faltos de clasificaciones de las métricas más representativas. Por ejemplo Robert Dunn[2] propone una clasificación basada en las medidas que podemos tomar de nuestro software durante su desarrollo (métricas para pruebas) o durante su periodo operacional (métricas para pruebas una vez el software ya se ha entregado), de su complejidad y finalmente de sus propiedades estructurales, dedicando una sección aparte a los modelos de fiabilidad. Sin embargo, Stephen Kan [1] prefiere hacer una clasificación de las métricas en función de la calidad del producto, de sus procesos d y de su mantenimiento.
Ninguna de ellas se traduce directamente en todas las características que hemos visto en el apartado anterior, pero proveen una evidencia sin discusión de la calidad del software. La clasificación que hemos elaborado se encuentra a medio camino de las que hemos mencionado:

1. Métricas intrínsecas al producto.

Normalmente la calidad intrínseca del producto viene relacionada con el número de fallos o defectos funcionales del software, pero también por su complejidad o por el tiempo que puede funcionar sin fallar. Veamos las métricas que identificamos dentro del grupo de intrínsecas del producto:

- Tiempo útil sin fallos[6]. (MTT[7]). Exactamente mide el tiempo entre los fallos. Es especialmente relevante para la estimación de costes en la fase de mantenimiento y determinar la fiabilidad actual del software. Un producto con un bajo MTT será poco usable, por lo tanto también nos ayuda a determinar la usabilidad de un producto software.
- Ratios de fallos. Podemos observar diferentes ratios de defectos o fallos:
  - Fallos por unidad de tiempo, bajo circunstancias operacionales.
  - Incidencias operacionales por unidad de tiempo operacional. O lo que es lo mismo, número de errores que han aparecido cuando el sistema esta realizando alguna operación o funcionalidad y no en "stand-by".
  - Ratio de defectos en función de LOC[8]. El problema actual es que el concepto de LOC puede variar en función del autor:

    - Contar solo líneas de código ejecutables.
    - Contar líneas de código ejecutables y definición de los datos.
    - Contar líneas de código ejecutables, definición de los datos y comentarios.
    - Contar líneas físicas, las que podemos ver en una pantalla.
    - Contar líneas que terminan con un delimitador lógico. Por ejemplo, ";".

    Por tanto el ratio de defectos vendrá determinado por el número de errores detectados entre el LOC que creamos más conveniente.

  - Ratio de defectos en función de puntos de función. Antes hemos hablado de expresar el ratio de errores en función del LOC, utilizar puntos de función no es mas que otra manera de estimar el tamaño de nuestro producto software y obtener otro ratio de fallos. Si definimos una función como una colección de sentencias ejecutables que realizan cierta tarea, junto con la declaración formal de los parámetros y variables locales que manipularán estas sentencias (Conte 1986), el ratio de fallos se indexa al número de funciones que el software provee.

- Complejidad. Diferentes técnicas han sido desarrolladas para calcular la complejidad de un producto software, entre ellas podemos encontrar Cyclomatic complexity de Thomas McCabe[6] y los cálculos de knot [7]. También se han llevado a cabo estudios para determinar la complejidad de los datos y los diferentes flujos en los que se mueve.
  Otro punto interesante es el cálculo de la complejidad del diseño, que prácticamente se ha llevado a cabo a partir de la modularidad[9] y de una métrica conocida como estabilidad del diseño. Esta métrica fue determinada por Yau y Colofello[8] a partir del concepto del efecto ripple. El efecto ripple no es mas

---

6 Un error es un fallo humano que provoca un software incorrecto. Un defecto es una anomalía en un producto. Un fallo ocurre cuando una unidad funcional, relacionada a un sistema software, no puede llevar a cabo una función requerida. Un fallo del software es la materialización de un error.
A lo largo del texto podremos encontrarnos indistinguiblemente los términos fallos o defectos del software para referirnos al mismo concepto.

7 Del ingles Mean Time to Failure

8 Del ingles, Lines Of Code, líneas de código.

9 La modularidad describe el grado de independencia entre diferentes partes estructurales de una mismo programa.

que el efecto que sufren los módulos de software cuando uno que esta relacionado con ellos es modificado.

2. *Métricas de los procesos.*

Estas métricas no están tan bien definidas como las métricas que podemos encontrar para los productos finales y su práctica puede variar enormemente según los desarrolladores de software. Por un lado tenemos métricas que indican un seguimiento de los diferentes errores que van apareciendo en ka máquina de pruebas y por el otro, métricas bien establecidas en cada unas de las fases del desarrollo del software. Veamos algunas de las métricas más utilizadas:

- Ratio de fallos en la máquina de pruebas. Cuando hablamos de máquina de pruebas nos estamos refiriendo a las pruebas que hacemos justo después de integrar el código a la librería del sistema o con el resto del código funcional. Esta métrica esta relacionada con el ratio de fallos del producto software y normalmente son correlativas, si el ratio en la máquina de pruebas aumenta puede ser debido a que se ha ha estado introduciendo código erróneo o a que se han añadido nuevas pruebas.
- Patrón de variaciones de los errores en la máquina de pruebas. Aquí podemos observar las diferentes tendencias de aparición de errores y al mismo tiempo la calidad del software en cada instante de tiempo reflejado en los patrones. Realmente estamos utilizando tres métricas aquí:
  - La aparición de errores durante la fase de pruebas por intérvalo de tiempo.
  - El patrón de aparición de errores verificados. Es decir de errores que hemos comprobado que realmente son errores y no falsos errores. Este es el patrón al que hacemos referencia al principio.
  - La tendencia o patrón de el número de errores por corregir a lo largo del tiempo. Ninguna compañia puede solucionar todos los errores al instante, por lo que se preparan listados o "backlogs" que indican que errores ir corrigiendo. Si esta lista de errores es muy grande al final del ciclo de desarrollo, la fiabilidad del sistema puede verse compromete y por ende, su calidad.
- Patrón de corrección de errores, orientado a fases. Es una extensión de la métrica que hemos visto con anterioridad, el ratio de fallos. En esta caso es necesario realizar un seguimiento de los errores en todas las fases del desarrollo del software, incluyendo la revisión del diseño, inspecciones de código y verificaciones formales después de las pruebas.
- Efectividad de la eliminación de los errores. Tal y como podemos encontrar en Metrics and Models in Software Quality Engineering[1], la eficiencia o efectividad de eliminación de errores se expresa como sigue:

$$DRE = (\text{errores eliminados en la fase de desarrollo} / \text{errores latentes en el producto}^{10}) \times 100$$

3. *Métricas mantenimiento del producto acabado.*

La fase de mantenimiento de todo producto se inicia cuando el producto se ha finalizado y se libera al mercado. Durante esta fase lo que se suele hacer es arreglar los errores detectados, ya sea por los clientes o por nosotros mismos, cuanto antes posible y de la mejor manera posible. Estas son las métricas más significativas:

---

10 Normalmente es un valor aproximado calculado como el número de errores durante la fase + errores encontrado después de la fase

- Número de errores por fijar y el indice de corrección (BMI[11]). Ambas métricas son especialmente útiles en la fase de mantenimiento, ya que nos permite controlar el rimto y evolución de la calidad de nuestro software. La primera métrica es simplemente un contador del número de errores que quedan por resolver a final de cada mes o de cada semana. Y la segunda es el porcentaje de errores resueltos respecto a los errores nuevos que se han detectado en un mes.

  BMI = (Nº de problemas resueltos en el mes / Nº de problemas detectados en el mes) x 100

- Tiempo de resolución de errores y tiempo de respuesta. Ambas métricas son muy importantes a la hora de incrementar la satisfacción del cliente. Mientras que la primera métrica hace referencia al tiempo que tardamos en solucionar un error, desde que se abre el expediente hasta que se cierra como solucionado, la segunda se centra en el tiempo que somos capaces de ponernos en contacto con el cliente y fijar un tiempo de resolución que obviamente hemos de cumplir.
- Porcentajes de previsiones incumplidas. Cada vez que establecemos una fecha para la cual un error estará solventado y no cumplimos esta previsión, entonces la consideramos como previsión incumplida o como dicen en ingles "a delinquent fix". El porcentaje se calcula de la siguiente manera:

  Delinquent fixes = (Número de correcciones que exceden tiempo previsto/
  Número de correciones hechas a tiempo) x 100

- Número de correcciones erróneas. Esta métrica hace referencia a aquellas correcciones que son defectuosas, es decir que corrigen el problema inicial pero introducen nuevos problemas o que simplemente ni corrigen el problema original.

4. Métricas satisfacción del cliente.

La satisfacción del cliente se suele medir a través de encuestas que requieren respuestas en la siguiente escala: 1-Muy satisfecho. 2-Satisfecho. 3-Neutral. 4-Poco satisfecho. 5-Nada satisfecho.
Las dimensiones por las cuales son cuestionados los clientes varían según la empresa (ver primer apartado, Qué es la calidad del software), pero fundamentalmente y basándonos en estas encuestas, podemos determinar las siguientes métricas, entre otras:

- Porcentaje de clientes satisfechos. Son aquellos que están satisfechos y muy satisfechos.
- Porcentaje de clientes poco satisfechos.
- Porcentaje de no satisfechos. Incluye neutrales, poco satisfechos y nada satisfechos.

11 Del ingles Backlog Management Index

Como asegurar la calidad

Uno de los primeros intentos por gestionar la calidad del software vino de la mano de la "Naval Air Systems Command", en 1985, en un intento de emular el estilo japonés en la mejora de la calidad. Se acuño el término Total Quality Management (TQM), gestión total de la calidad. A grandes rasgos, representaba un estilo de gestión orientado a lograr éxitos a largo plazo, a través de la unión de la calidad y la satisfacción del cliente. Un elemento básico era la creación de una cultura en la cual todos los miembros de la organización participan en la mejora de los procesos, productos y servicios.

Por parte de la comunidad europea se estableció la ISO 9000 como el estándar de gestión de la calidad, HP estableció un TQM que llamó Hewlett-Packard's Total Quality Control (TQC) (control de calidad total), IBM Market Driven Quality (Calidad orientada al mecado), Motorola Six Sigma Strategy ,...
Pero a pesar de todas estas diferentes implementaciones, un sistema TQM se caracteriza por :

- Centrado en el cliente. el objetivo es conseguir la satisfacción total del cliente. Incluye el estudio de las necesidades y demandas de los clientes, recogiendo los requisitos de los clientes y midiendo y gestionando la satisfacción del cliente, de la cual hemos hablado en la primera sección anterior.
- Procesos. EL objetivo es reducir las fluctuaciones en los procesos y alcanzar la mejora continua de los mismos. Se entiende que a través de la mejora de los procesos alcanzaremos una mayor calidad de nuestros productos.
- Calidad basada en las persona. El objetivo es crear una cultura de calidad a nivel de las personas que conforman la compañía.
- Métricas, modelos, medidas y análisis. El objetivo es dirigir la mejora continua de todos los parámetros de calidad, a través de un sistema de medidas orientado a los objetivos.

Algunas organizaciones propusieron diferentes frameworks para mejorar la calidad, que venían a corroborar la filosofía de TQM. Entre ellos podemos encontrar Capability Maturity Model del Software Engineering Institut (SEI), Lean Enterprise Management o Plan-Do-Check-Act.

Software Quality Assurance

O lo que es lo mismo, garantía de la calidad del software, se define como una planificada y sistemática aproximación a la evaluación de la calidad y a la adherencia de los estándares de los productos software, procesos y procedimientos. SQA incluye los procesos para garantizar que los estándares y procedimientos son establecidos y seguidos a través de las diferentes fases del desarrollo del software. Simplemente lo que estamos diciendo es que SQA nos garantiza que vamos a introducir en cada una de las fases del desarrollo de nuestro producto software elementos estandarizados para gestionar, controlar y garantizar la calidad de nuestro producto final. El cumplimiento de los estándares y procedimientos es evaluado mediante la monitorización de los procesos, la evaluación del producto y auditorias.
No entraremos en mayor detalle con respecto a SQA, si quieren profundizar más en el tema dirijanse a los siguientes recursos [2][9][10].

Conclusiones

La calidad de nuestro software es importante y hemos de saber como medirla, gestionarla, asegurarla y mejorarla. Para medir la calidad del software hemos visto sus principales atributos fiabilidad, usabilidad, mantenibilidad y adaptabilidad, y hemos reflexionado sobre la percepción de la calidad y la importancia que tiene la valoración de nuestro clientes y su satisfacción con nuestro producto. Con tal de mejorar la calidad de nuestro software hemos de ser capaces de medirla primero y de este modo tener unos indicadores de nuestra evolución, con este fin hemos determinado métricas que nos permiten valorar los principales atributos de calidad y obviamente, la satisfacción del cliente.
Para que todo esto tenga sentido, hemos de ser capaces de gestionar la calidad y asegurarla, con este propósito aparecieron técnicas de gestión de la calidad que a su vez originaron frameworks y estándares que garantizan una calidad basada es unos principios que ellos establecen y que todos acordamos como válidos.

Referencias

[1] Stephen H.Kan. Metrics and Models in Software Quality Engineering 2002

[2] Robert H. Dunn. Software Quality, Concepts and Plans 1990

[3] Glossary of Software Engineering Laboratory Terms (SEL-82-005) National Aeronautics and Space Administration (December 1982)

[4] IEEE Standard Glossary of Software Engineering Terminology. IEEE Std. 729-1983 (February 1983)

[5]Draft Military Standard, Software Quality Evaluation (DoD-STD-2168) Department of Defense (April 1985)

[6] Thomas McCabe, "A Complexity Mesure" IEEE Trans. Software Eng., Vol. SE-2 (December 1976) pp.308-320

[7]M.Woodward. "A Mesure of Control Flow Complexity in Program Text", IEEE Trans Software Eng., Vol SE-5 (January 1979) pp. 45-50

[8]J.Colofello and S.Yau, "design Stability Measures for Software Maintenance", IEEE Trans. Software Eng. Vol. SE-11 (September 1985) pp. 849-856

[9]The Well -http://www.well.com/user/vision/sqa.html

[10]Software Assurance GuideBook and Standard - http://satc.gsfc.nasa.gov/assure/assurepage.html

5. Metodologías

En este capítulo veremos que son las metodologías del software, como, cuando y porque surgieron. Desde una perspectiva histórica entenderemos su origen y estudiaremos las principales metodologías tradicionales. Procesos, roles, ámbito de uso y experiencias, son los apartados fundamentales que caracterizaremos de cada una de las metodologías ágiles, del mismo modo que hemos hecho anteriormente con las tradicionales. Al finalizar este capítulo tendremos un mayor conocimiento de las metodologías y estaremos preparados para realizar una comparativa de las metodologías seleccionadas.

Contenidos de la sección

Qué es una metodología de desarrollo

"*Una metodología es una colección de procedimientos, técnicas, herramientas y documentos auxiliares que ayudan a los desarrolladores de software en sus esfuerzos por implementar nuevos sistemas de información. Una metodología esta formada por fases, cada una de las cuales se puede dividir en sub-fases, que guiarán a los desarrolladores de sistemas a elegir las técnicas mas apropiadas en cada momento del proyecto y también a planificarlo, gestionarlo, controlarlo y evaluarlo.*"[1]

Avison y Fitzgerald, nos presentan una descripción de las metodologías de desarrollo muy clara y que destaca sus principales componentes, fases, herramientas y técnicas. Pero una metodología es algo más que una colección, casi siempre se basa en un filosofía, distinguiéndose de esta manera de los métodos o de las simples recetas, que nos marcan unos pasos a seguir y ya esta. De esta manera podemos observar que las diferentes metodologías que existen, se diferencian ya sea bien por las diferentes técnicas y herramientas que utilizan en cada fase, el mismo contenido de la fase o en la base de su filosofía, centrándose en realizar un enfoque científico u orientándolo a las personas, intentando automatizar al máximo las tareas,…

Conceptos sobre metodologías

Una técnica es según la definición de la Real Academia de la lengua, un conjunto de procedimientos y recursos de los cuales se sirve una ciencia o un arte. En nuestro caso, los recursos serán usualmente herramientas y los procedimientos, unas reglas determinadas con las que se intenta llevar a cabo un conjunto de actividades que forman parte del proceso de creación del producto software, normalmente en una o más fases. Podemos encontrarnos que muchas veces se utiliza el término técnica, para identificar una metodología, debemos evitar caer en este error, ya que una metodología puede utilizar diferentes técnicas. Como ejemplo de técnica podemos mencionar el modelado de entidades relacionales, normalización, etc.

Una herramienta es un conjunto de instrumentos que se utilizan para desempeñar un oficio o un trabajo determinado. En nuestro caso, podemos decir que una herramienta es el instrumento mediante el cual podemos llevar a cabo las técnicas determinadas por una metodología de desarrollo del software. Este instrumento por lo general será un producto software. Ejemplo de herramientas de desarrollo de software son las CASE tools, herramientas de modelado, repositorios de software, etc.

Un método es según la Real Academia de la Lengua, el modo de obrar o proceder, hábito o costumbre que cada uno tiene y observa. El método nos indicará las reglas a seguir, unos pasos que nos servirán para desarrollar una actividad en concreto. Cada técnica suele venir acompañada de unos métodos que nos indican como aplicarla a través de sus herramientas asociadas.

Un proceso de desarrollo del software es el conjunto de actividades que se llevan a cabo para producir un producto software. Este proceso se puede realizar de infinitas formas y produciendo siempre resultados dispares, las metodologías intentan marcar un camino común a través del cual poder producir software con éxito. El proceso de desarrollo del software esta compuesto por actividades, pero también por subprocesos, el control de estas actividades y su verificación, llevan implícito el concepto de calidad. Muchas metodologías incluyen dentro de sus técnicas el control de los procesos de desarrollo, con el fin de garantizar una solución software de calidad. Normalmente lo hacen como si el producto software fuese un producto industrial más y certifican sus procesos contra entidades certificadoras (ISO), las cuales enumeran las buenas practicas que se deben seguir. En el capítulo, *Calidad del Software*, hemos enumerado algunas metodologías basadas en la mejora de procesos y hablado más extensamente de los factores que se tienen en cuenta en la calidad del software.

Un modelo de procesos es una representación del mundo real, que captura el estado de actual de las actividades para guiar, reforzar o automatizar partes de la producción de los procesos[4]. Veamos los diferentes modelos de procesos que nos podemos encontrar dentro del mundo del desarrollo del software:

- Modelo secuencial. Representado por metodologías tan famosas como Waterfall. Se inicia con un completo análisis de los requisitos de los usuarios. Después de meses de intensa interacción con el usuario y los clientes, los ingenieros determinan un conjunto de características, requisitos funcionales y no funcionales. Toda esta información es bien documentada para la siguiente fase, diseño, donde los ingenieros colaboran entre ellos para crear una arquitectura óptima del sistema. En el siguiente paso, los programadores implementan el diseño y finalmente, el completado y perfecto sistema es probado y enviado. El modelo waterfall resuelve el problema de los requisitos variables, congelándolos, sin permitir los cambios. Simplemente lo enunciamos por ser una referencia en los modelos de procesos formales.

- Modelos iterativos e incrementales. Ambos rompen el ciclo de desarrollo y repiten el modelo waterfall en cada una de las partes en las que lo dividen.

    - Desarrollo incremental. Su principal objetivo es reducir el tiempo de desarrollo, dividiendo el proyecto en intervalos incrementales superpuestos. Del mismo modo que con el modelo waterfall, todos los requisitos se analizan antes de empezar a desarrollar, sin embargo, los requisitos se dividen en "incrementos" independientemente funcionales. El desarrollo de cada incremento se puede solapar, ahorrando tiempo gracias al desarrollo concurrente "multitarea" a lo largo del proyecto.

    - Desarrollo iterativo. A diferencia del modelo incremental se centra más en capturar mejor los requisitos cambiantes y la gestión de los riesgos. En el desarrollo iterativo se rompe el proyecto en iteraciones de diferente longitud, cada una de ellas produciendo un producto completo y entregable. La primera iteración empieza con una versión muy simple, y cada una de las siguientes iteraciones añade un conjunto de funcionalidades. Cada una de las partes, iteraciones, siguen en si el modelo waterfall, empezando con el análisis, seguido del diseño, implementación y finalmente las pruebas. El desarrollo iterativo funciona bastante bien en ambientes cambiables, ya que los requisitos son estáticos para cada iteración.

- Modelo en Espiral. Comprende las mejores características de ciclo de vida clásico y el prototipado (desarrollo iterativo). Además, incluye el análisis de alternativas, identificación y reducción de riesgos. (fig 1)]

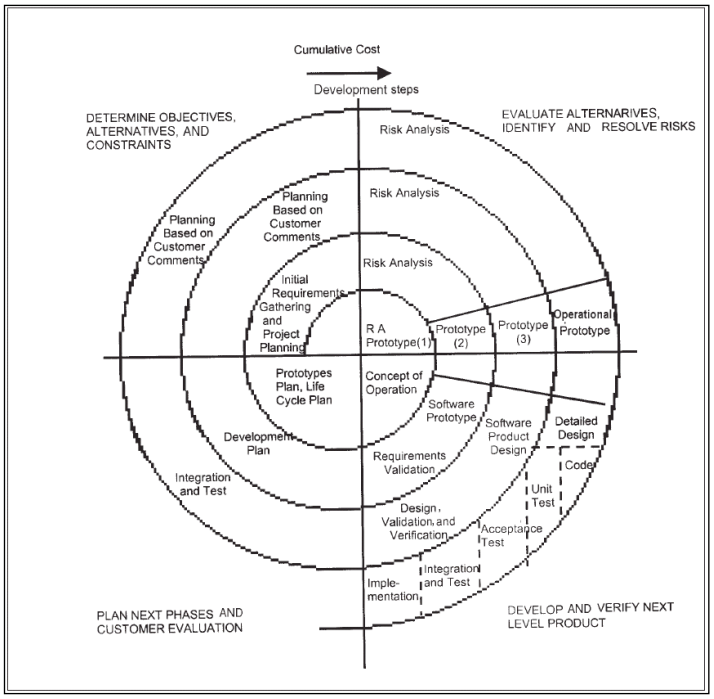


fig 1

El modelo en espiral y el iterativo, proporcionan un nivel de agilidad muy por encima de Waterfall, pero muchos usuarios consideraron que aún no eran suficientes para el cambiante mundo de negocios. Unas largas fases de planificación y análisis, así como un gran entusiasmo por el exceso de documentación, desvían a los proyectos que utilizan técnicas iterativas de ser completamente ágiles, en comparación con los métodos actuales. Otros modelos menos relevantes para las metodologías ágiles son V-Model, W-Model, X-Model, RAD y Orientado a Objetos, etc . Podemos encontrar una descripción más extensa de estos modelos en el artículo de E.Georgiadou *Software Process and Product Improvement: A Historical Perspective* [3].

Por tanto cuando hablemos de metodologías hemos de tener en cuenta todo lo que va implícito dentro de ellas, sus métodos, sus modelos, sus herramientas, su filosofía. Muchos autores cuando hablan de metodologías de desarrollo distinguen entre metodologías maduras e inmaduras, considerando a estas últimas todas aquellas difieran de los métodos formales y presentan ciertos signos de debilidad, ya sea en el producto final, como en el desarrollo del mismo.

Contexto Histórico

Para ser estrictamente rigurosos y mostrar un contexto histórico que enseñe correctamente el caldo de cultivo que provocó la aparición de las diferentes metodologías de desarrollo, podríamos remontarnos hasta Ada Lovelace, Babbage, Turing y compañía, en los inicios de la informática y los computadores, pero no será necesario remontarse tan atrás en el tiempo. Podemos simplemente comenzar con Dijkstra y su influyente articulo sobre la inconveniencia de utilizar la sentencia "Go To" en el código de nuestros programas (Go to Statement Considered Harmful) y la popularización de los "mini-ordenadores" junto con la archiconocida Ley de Moore[1].

Nos situamos a finales de los años 60, concretamente en 1968. Hasta entonces, los programadores no prestaban excesiva atención a su estilo de programación, dado los limitados recursos de espacio y velocidad de las primeras computadoras, sus mayores preocupaciones se referían a escribir código que ocupase poco y fuese muy eficiente. Los compiladores tampoco eran muy buenos y los programadores daban mayor prioridad a el uso de pequeños trucos, buscando generar el código más eficiente. Pero todo esto cambio cuando empezaron a proliferar "pequeños" y "baratos" equipos, que evolucionaban siguiendo la Ley de Moore . Esta situación cambió para siempre el orden de prioridad de los criterios que medían la efectividad de un programa, ordenadores más rápidos y con mayor almacenamiento no requerían aplicaciones centradas en optimizar el uso de la memoria y con un código muy eficiente. Con la aparición del popular System/360 de IBM y la generalización del uso de los lenguajes de alto nivel, se aseguraba que los programas fuesen portables y perdurables a lo largo del tiempo, hecho que provocó por primera vez en la historia, que los costes de desarrollo del software superasen a los del hardware. Este fue el inicio de la conocida crisis del software que aun hoy en día sigue vigente. Se establecieron unos criterios que marcasen el éxito del desarrollo del software, son los siguientes:

- El coste del desarrollo inicial debe ser relativamente bajo.
- El software debe ser fácil de mantener.
- El software debe de ser portable a nuevo hardware.
- El software debe hacer lo que el cliente quiere.

A pesar de todo, la mayoría de programadores continuaron desarrollando el software sin tenerlo muy en cuenta. Dijkstra planteó las ciencias de la computación como una rama de las matemáticas aplicadas y estableció las bases de la programación estructurada, la cual puede considerarse como una primitiva iniciación a las metodologías de desarrollo del software. A grandes rasgos, la programación estructurada se basa en:

- Desarrollo de programas mediante top-down, en contraposición a bottom-up.
- Utilizar un conjunto específico de reglas de programación. (Prohibido el uso de Go To)
- Seguir un conjunto de pasos formales para descomponer los problemas grandes (divide y vencerás)

A partir de que Dijkstra plantease su programación estructurada (a través de su libro A Discipline of Programming), empezaron a surgir lenguajes de programación influenciados por este, como fue Pascal, que no implementa la sentencia Go To y sigue las ideas de crear un código bien estructurado y legible. Pero Dijkstra no sólo inició el gran debate del "Go to", sino que provocó que a partir de su programación estructurada surgiese la idea del diseño y análisis estructurado, y lo que hoy conocemos como Ingeniería del Software. Concretamente en la conferencia de la NATO[2] de 1968, se reunieron

---

1 La ley de Moore fue enunciada por Gordon. E. Moore, co-fundador de Intel, el 19 de Abril de 1965. Esta ley predice que el número de transistores de un computador se duplicará cada 18 meses o 2 años aproximadamente.

2 North Atlantic Treaty Organization, su acrónimo en español, francés y portugués es OTAN, Organización del Tratado del Atlántico Norte.

para discutir los problemas que acuciaban el desarrollo del software y se acuñó por primera vez el término de ingeniería del software, planteando una solución a porque los proyectos de software fracasaban. La solución que se buscó fueron unas pautas formales, de ingeniería, para solucionar los problemas que había, estas pautas se basaban en los dos pilares básicos en los que se sustentan todas las ingenierías:

1. Gestión predictiva. Antes de abordar el proyecto vamos a tener claro que es lo que vamos a hacer, tomar requisitos, hacer un plan y con esto calcular lo que nos va a costar en dinero y esfuerzo. Después gestionar los riesgos para que el plan se cumpla.

2. Principios de calidad. La calidad del resultado depende sobretodo de la calidad de los procesos con los que se hace ese producto.

Estas son las bases de las metodologías formales.

En resumen, el desplome del precio del hardware había puesto el software en el punto de mira y los managers estaban desesperados por controlar el amplio y complejo proceso de desarrollo del software. Los clientes estaban empezando a molestarse por las grandes sumas gastadas y los pobres resultados obtenidos en los proyectos de desarrollo de software, que cada vez eran más grandes y costosos, algo se tenía que hacer. Era el momento ideal para la primera tesis, las metodologías de desarrollo del software.

Cronología de las metodologías

En este apartado presentamos la aparición de las diferentes metodologías a lo largo del tiempo y para ello hemos seguido la clasificación temporal que establecieron Avison y Fitzgerald[1][3]. Esta clasificación establece cuatro periodos de tiempos:

1. Periodo de pre-metodologías.
2. Primeras metodologías.
3. La era de las metodologías.
4. Era post-metodológica.

La primera era o periodo que consideramos es de pre-metodologías. Este periodo de tiempo se caracteriza por la ausencia de metodologías en el desarrollo del software y se sitúa temporalmente en los años 50 y 60 del siglo XX. Eran los inicios de las grandes computadoras, las cuales se asociaron a aplicaciones científicas y militares. Poco a poco se fueron introduciendo en entornos empresariales, de la mano de pequeñas guías de usuario que intentaban facilitar su traumático uso comercial. Estas aplicaciones empresariales realizaban reportes de ventas, guardaban información de los usuarios, ficheros…
Estos sistemas eran implementados por programadores, que no eran necesariamente buenos comunicadores, dificultando la comunicación con los usuarios. En esta etapa, los proyectos software se planteaban como pequeños ejercicios, los cuales se llevaban a cabo en cortos plazos de tiempo y no se pensaba en soluciones a largo plazo y bien planeadas. Los usuarios raramente estaban satisfechos con la solución presentada, la documentación era escasa (si existía), los cambios cada vez requerían de mayor tiempo a la vez que la aplicación se hacia más compleja y los programadores eran piezas cotizadas, ya que eran los únicos que sabían como funcionaban internamente estas aplicaciones.
Hubo una reacción a esta situación, se dieron cuenta de que era necesario dedicar más tiempo al análisis y diseño de la aplicación. Aparecieron las figuras de los analistas de sistemas y operadores, ademas del ya existente perfil de programador. Los operadores controlaban el funcionamiento de las máquinas y los analistas de sistemas actuaban en un rol intermedio entre los usuarios y el programador, a veces se

3 También consideramos seguir la clasificación temporal que E.Georgiadou describe en Software Process Improvement: A Historical Perspective [3], pero esta clasifiación resultaba algo confusa y no recogía los últimos 8 años.

distinguían dos tipos de analistas, el analista de negocio que exclusivamente se encargaba de recoger las necesidades del sistema y comunicárselas a un segundo analista, el analista técnico, que diseñaba la solución y se la pasaba al programador.

La segunda reacción fue la aparición de las metodologías, tal y como hemos comentado en el apartado anterior. Podemos considerar al periodo comprendido entre 1970 y 1980, como la juventud de las metodologías, el periodo que hemos identificado como primeras metodologías. Fue en 1971 cuando Daniel y Yeats describieron completamente "el modelo waterfall", la primera metodología basada en el ciclo de vida de desarrollo de un sistema software. Esta metodología esta formada por un conjunto de etapas que debían ser seguidas secuencialmente, una fase debe estar completamente finalizada para que comience la siguiente (de ahí el término de waterfall o cascada) y cada fase tiene asociadas unas salidas o entregables para ser considerada finalizada. Esta metodología fue la primera y la abanderada de las que actualmente conocemos como metodologías tradicionales o formales, que basan principalmente su potencia en la ingeniería de procesos.
A pesar de todo, SDLC (Systems Development Life Cycle) o waterfall, tiene serias limitaciones como la imposibilidad de encontrar las verdaderas necesidades del cliente (debido a que se centra en la mejora de la eficiencia tecnológica), inestabilidad(debida a los requisitos variables del cliente), poco flexible, , insatisfacción de los clientes, exceso de documentación y la facilidad de desviarse del tiempo planificado (debidos a los intentos por cambiar el sistema para que c incluya los nuevos requisitos del sistema).

Un gran número de nuevos enfoques surgieron como respuesta a las limitaciones de SDLC, dando lugar al inicio de la mencionada era de las metodologías. Podemos observar dos corrientes diferentes a seguir en este periodo de tiempo, una que decide mejorar el modelo waterfall y otra que decide hacer algo diferente. Hemos de destacar que el modelo waterfall se mejoró a través de la incorporación de técnicas y herramientas que se fueron desarrollando a lo largo de los años setenta. Entre estas técnicas encontramos el modelado de entidades relacionales, normalización, diagramas de flujo de datos, diagramas de actividad, diagramas estructurados y entidades de ciclos de vida. Destacamos herramientas de gestión de proyectos software, repositorios de código, de modelado y las complejas herramientas de CASE (Computer Assisted Software Engineering).
Las metodologías que podemos considerar como actualizaciones o versiones mejoradas del modelo Waterfall son Merise, SSADM o Yourdon System Method. Estas metodologías integran las técnicas y herramientas mencionadas anteriormente, actualizando y mejorando el modelo del ciclo de vida.
Enfoques alternativos fueron desarrollados a lo largo de los años ochenta, Avison[1] realiza una clasificación muy intuitiva y que ayuda a agrupar la mayoría de las metodologías que surgirían en esta época, la clasificación es la siguiente:

- Sistemas. Abanderado por la metodología desarrollada por Checkland, Soft Systems Methodologies (SSM), plantean el uso de técnicas cualitativas que pueden ser usadas para aplicar un pensamiento sistemático (Systems Thinking) a situaciones no sistemáticas. De esta manera quieren gestionar situaciones problemáticas con un alto grado social, político y con una actividad humana como componente principal.

- Estratégicas. Incluyen aquellas metodologías que ponen mayor énfasis en la planificación previa al desarrollo del software y a la necesidad de desarrollar una estrategia para alcanzar los objetivos de negocio. La panificación de sistemas de negocio (Business Systems Planning) de IBM es un ejemplo de este enfoque y los procesos de reingeniería de negocio se consideran un enfoque de desarrollo basado en la estrategia.

- Participativas. La característica principal es la participación de los usuarios. Se involucra a los usuarios y otros posibles interesados en el sistema (Stakeholders) en las fases de análisis, diseño e implementación, de este modo obtienes el beneplácito de los usuarios a la implementación que vas a realizar. Una metodología representativa de este enfoque es ETHICS.

- Prototipadas. Este enfoque se caracteriza por el uso de prototipos, que no son más que una aproximación o representación del sistema, que permiten a los usuarios visualizar y responder en función de la implementación realizada. Implementado en primera instancia prototipos, los analistas pueden mostrar a los usuarios las entradas, los estadios intermedios y las salidas del sistema, de una manera que los usuarios entienden. Rapid Application Development (RAD) es un ejemplo que utiliza prototipado.

- Estructuradas. Principalmente es la extensión de los conceptos de la programación estructurada, extendidos al análisis y el diseño, y técnicas que habilitan el análisis descendente (top-down) y representación compleja de procesos. Este enfoque suele sobre utilizar técnicas como árboles decisionales, tablas, diagramas de flujos de datos y herramientas como repositorios.

- Análisis de los datos. A diferencia del enfoque estructurado, el cual se centra en los procesos, aquí nos centramos en entender y documentar los datos o información. Es decir que entendemos como el elemento principal de desarrollo lo datos y de esta manera, la técnica ampliamente utilizada es el modelado de entidades relacionales. La ingeniería de la información (Information Engineering) utiliza un enfoque basado en los datos.

Los primeros años de la década de los noventa aún se pueden considerar dentro de la era metodológica y presentaron una nueva oleada de metodologías, que clasificamos en los siguientes grupos:

- Orientadas a objetos. Yourdon expuso una metodología basada en la orientación a objetos arguyendo que este enfoque es más natural que no los presentados anteriormente sobre los datos o los orientados a los procesos (estructurado), a la par que unifica los procesos de desarrollo de los sistemas de información. Otros beneficios presentes de esta solución son, la facilidad presente para la reutilización de código, la habilidad de enfrentarse a los problemas con mayores garantías de éxito, debido en gran parte al grado de entendimiento que aporta este enfoque sobre las situaciones problemáticas. Otro beneficio más es la mejora de la relación entre analistas y usuarios, ya que este enfoque no es orientado a los ordenadores, sino a los objetos, que representan entidades del mundo real.

- Desarrollo incremental o evolutivo. Este enfoque se caracteriza por incluir las maneras de hacer del enfoque de prototipado y por desarrollar software siempre de manera incremental, es decir que se construye siempre a partir de versiones anteriores y nunca desde cero. Promete una reducción significativa del tiempo de desarrollo de un producto software, a su vez que asegura que a partir del conocimiento adquirido durante su desarrollo se pueden captar las variaciones de los requisitos. El sistema que se desarrolla se suele dividir en diferentes componentes que pueden ser desarrolladas separadamente. La metodología DSDM incluye este enfoque incremental y muchos proyectos open source utilizan la característica de desarrollar mediante componentes independientes.

- Específicas. Algunas metodologías han sido concebidas exclusivamente para un tipo de aplicaciones, estas metodologías pueden considerarse como metodologías con un propósito determinado, encontramos las siguientes metodologías:

    - Welti, para el desarrollo de aplicaciones ERP.
    - CommonKADS, para el desarrollo de aplicaciones de gestión del conocimiento.
    - Process Innovation (Innovación de procesos), para el desarrollo de aplicaciones de reingeniería de procesos de negocio.
    - Renaissance, metodología que se basa en la técnica de reverse engineering para sistemas heredados.
    - WISDM, para el desarrollo de aplicaciones basadas en el Web.

Muchos usuarios no estaban del todo satisfechos con las metodologías que utilizaron, ya fuese el modelo waterfall o alguna de sus alternativas. La mayoría de las metodologías habían sido diseñadas para situaciones ideales y todos sabemos que eso no existe, cada situación, cada proyecto es distinto. Hemos de añadir, que muchos usuarios de las metodologías, esperaban encontrarse un guión bien definido que los guiase paso a paso y mediante un enfoque descendente (top-down) a lo largo del desarrollo del software. Esta situación ocasionalmente se produce, ya que normalmente hemos de adaptar ese guión, omitiendo fases e incluso llevándolas a cabo en un orden diferente al especificado. De la misma forma, muchas técnicas y herramientas podían ser usadas o no, según las circunstancias.

La respuesta más lógica a esta situación es el planteamiento de un enfoque más relajado, menos prescriptivo, un enfoque contingente. En este nuevo enfoque se presenta una estructura, pero las fases, herramientas, técnicas y demás parámetros, se dejan a elección del usuario para usarlos o no, dependiendo de cada situación. Multiview es un ejemplo de metodología que sigue un enfoque contingente.

El éxito o fracaso del desarrollo del software no se puede atribuir únicamente al uso o desuso de las metodologías, pero a pesar de esto, observamos que a finales de los años 90 surge una corriente caracterizada por una seria reevaluación de los supuestos beneficios de las metodologías, incluso se observa una violenta reacción en contra de las metodologías, juntamente con un conjunto de diversos enfoques no metodológicos, que dan lugar a lo que podemos denominar una era "postmetodológica".

Existen múltiples causas que podrían explicar esta reacción adversa a la utilización de metodologías, pero podríamos afirmar que una de las más importantes es el desencanto provocado por el no cumplimiento de las tan altas expectativas mostradas inicialmente. De todos modos, sería injusto decir que las metodologías han sido un fracaso estrepitoso, es más correcto afirmar que las metodologías no han solucionado todos los problemas que se suponía que debían solucionar.

Veamos algunas de las razones que Fitzgerald y Avison [1]presentan como las causantes de que las empresas se planteen el uso de metodologías:

- Productividad. Las metodologías no supusieron el incremento de productividad que inicialmente habían prometido
- Complejidad. Fueron criticadas por ser excesivamente complejas.
- Habilidades. Metodologías requerían habilidades específicas para su uso.
- Herramientas. Las herramientas que proponían las metodologías eran difíciles de utilizar, caras y no generaban los suficientes beneficios.
- No contingente. No se adaptaban a las necesidades del proyecto.
- Enfoque unidimensional. Las metodologías suelen utilizar un único enfoque para el desarrollo de los proyectos.
- Inflexible. Se muestran inflexibles respecto a los cambios.
- Supuestos no válidos. Como que los requisitos no variaran a lo largo del desarrollo de proyecto.
- Desplazamiento del objetivo. Cuando nos centramos en seguir una metodología al pie de la letra, sin pensar que quizás no es la manera más adecuada para nuestra situación. Esta situación provoca la desaparición del pensamiento creativo.
- Poco orientadas a las personas y al entorno.
- Dificultades para adoptar una metodología.

Tal y como hemos ido viendo a lo largo de esta cronología de las metodologías, siempre que algo no funcionaba, surgía un movimiento o reacción que trataba de solventar los problemas, veamos que alternativas se proponen en función de las necesidades de cada empresa, la mentalidad de los usuarios, etc. Las alternativas que contemplamos son:

- Desarrollo externo. Muchas compañías han decidido no realizar más proyectos software "in-house" y comprar todos sus requisitos en forma de paquetes. De aquí que soluciones ERP se hayan extendido y hayan tenido tanto éxito.

- Mejora continua. Abogan por la continua mejora de las metodologías, que evidentemente siempre podrán ser mejor.

- Desarrollo ad-hoc y contingencia. El desarrollo ad-hoc es como un retorno a la era que hemos identificado como pre-era de las metodologías, donde no se seguía ningún tipo de metodología formal. El enfoque es el siguiente, cualquier cosa que el desarrollador entienda y sienta, funcionará. Evidentemente se basa en la experiencia y habilidad de los desarrolladores.

- El desarrollo contingente o flexible, es el que sigue un enfoque contingente, como hemos comentado brevemente unas líneas atrás.

- Desarrollo ágil. El manifiesto ágil sugiere un enfoque orientado a la participación de los usuarios y clientes, más que hacia los procesos y herramientas, trabajando más en el software y menos en la documentación, colaborando mas con los clientes en vez de estar negociando y respondiendo a los cambios sacrificando el plan de trabajo si es necesario.

Necesidades de una metodología

Larry Trussell [3] propone los siguientes puntos primordiales que debe tener una metodología:

1. Visión del producto. Todo el mundo debe conocer lo que el equipo esta tratando de hacer, como debe de ser el producto final, las bases de la estrategia del producto y cuando el producto será entregado.
2. Vinculación con el cliente. La metodología se debe encargar de indicar la manera de gestionar el vínculo entre clientes, desarrolladores, especificación de requisitos y el personal de soporte.
3. Establecer un modelo de ciclo de vida. Un modelo de ciclo de vida como pueden ser el iterativo, secuencial o waterfall, etc. De esta manera se establecen los pasos en el proceso de desarrollo y se pueden ubicar los recursos adecuadamente.
4. Gestión de los requisitos. Nivel de detalle que deben tener los requisitos del producto, siendo recomendable cuanto más alto mejor.
5. Plan de desarrollo. Un documento con un plan para organizar los requisitos y cuestiones relacionadas con la calidad. Los ítems de este plan deben ser lo suficientemente detallados para que los desarrolladores de software puedan desarrollar sus tareas de codificación de un modo no ambiguo. Un proceso para añadir y modificar el documento se debe especificar, como mínimo para mantener un histórico de los cambios.
6. Integración del proyecto. Una metodología de desarrollo debe conducir a una organización a determinar como se integrará el producto fabricado con los existentes y futuros productos de la compañía.
7. Medidas de progreso del proyecto. Se considera un aspecto crítico en una metodología. Desarrolladores, manager y los altos cargos de la organización deben entender el progreso del desarrollo del equipo de desarrollo. Ellos deben conocer el estado actual del producto, así como una buena estimación del tiempo que resta para la finalización del proyecto.
8. Métricas para evaluar la calidad. El responsable de las release del producto depende de un proceso de para la medida de la calidad, el cual suele empezar en las primeras etapas de la planificación. Este proceso no es tan solo para encontrar fallos, produce indicadores de la robustez del producto y cuanto se aproxima el producto a las especificaciones iniciales.
9. Maneras de medir el riesgo. Un plan debe tener en cuenta los posibles problemas que pueden ocurrir durante el proceso de desarrollo, el impacto de estos problemas y que acciones deberían ser llevadas a cabo para solucionar o prevenir estos problemas. La gestión de los riesgos es la responsabilidad diaria de los project managers y desarrolladores.
10. Como gestionar los cambios. Nuevas ideas y problemas desembocan en cambios de diseño y especificación aún que ya hayamos empezado a implementar. Un plan debe contemplar estas sugerencias para introducirlas en el proyecto, debatirlas e implementarlas.
11. Establecer una línea de meta. La metodología de desarrollo debe forzar a una organización a especificar exactamente que esta siendo construido y que constituye el producto final. Todo el mundo en la organización debe tener un visión clara del producto final y entender que significa “terminado”.

Comparto con Larry la necesidad de establecer una visión inicial del producto, pero tal y como podremos observar más adelante en las metodologías ágiles como Scrum, existen parámetros determinantes de un documento visión que no se pueden fijar, como pueden ser el tiempo de entrega o el aspecto final del producto.
En referencia al segundo punto que propone, vinculación con el cliente, hemos de ir con cuidado, ya que es susceptible de entrar en conflicto con aspectos sociales y maneras de hacer de la empresa. Es uno de los puntos que debe mostrar mayor flexibilidad.
La propuesta de establecer un modelo de ciclo de vida es una posición tradicional de la definición y creación de metodologías. Estoy de acuerdo con que para poder planificar un proyecto, poder establecer fechas y estimar costes, una de las maneras mas utilizadas es utilizar un modelo de ciclo de vida. Pero hemos de observar cuan peligroso es intentar tratar un proyecto software siempre como

algo que se puede predecir a lo largo del tiempo, de todos modos considero útil incluir un modelo de ciclo de vida que sea lo más flexible posible y teniendo en cuenta todos sus riesgos asociados.
El autor propone la inclusión de una guía de gestión de los requisitos, la cual determine como de específicos o detallados deben de ser. Para mi sería mas adecuado entender por gestión de requisitos no solo su nivel de detalle, sino también la gestión de su ciclo de vida. Es decir, desde una perspectiva ágil, los requisitos siempre son bienvenidos, una metodología debería indicar de que manera gestionar los nuevos requisitos, como tratar modificaciones de los existentes y sobretodo como adecuarlos al plan de desarrollo o implementación, indicando en cada caso las posibles desviaciones tanto de tiempo como de recursos económicos y personales.
Establecer un plan de desarrollo es para mí un punto importante en el desarrollo del software, ya que orienta, gestiona y controla la evolución del proyecto. Del mismo modo, considero interesante la inclusión del punto integración del proyecto. Hace referencia a la necesidad de que una metodología oriente a sus usuarios a trabajar en la dirección de que su producto debe ser integrado en una organización, con otros productos y que posiblemente futuros productos serán también integrados.
Las medidas de progreso del proyecto son a mí entender, un factor imprescindible en toda metodología, indistintamente de su filosofía y enfoque, ya que es uno de los métodos más eficaces para el control de un proyecto. De todos modos también considero que es una medida complicada de estimar y en función de cómo se haga puede ocasionar grandes perjuicios al proyecto.
Otro punto que considero muy importante en todo proyecto software y que debe incluir toda metodología es de las métricas de evaluación de la calidad. De la misma manera que con las medidas de progreso, es un arma de doble filo. La selección de métricas eficaces no es trivial, ni encontrar el equilibrio entre un exceso de control y una ausencia preocupante de estas. Tal y como el autor plantea las métricas de calidad, no deja claro si se consideran métricas que establezcan la calidad de los procesos y no solo del producto final. Métricas para la calidad de los procesos, que a su vez garanticen la calidad del producto que se esta elaborando, podemos encontrarlas especificadas en la serie de ISO 9000 o concretamente aplicadas en CMMI.
La gestión de riesgos es sin duda un trabajo que debe de ser llevado a cabo diariamente, no hay bolas mágicas que puedan predecir imprevistos, cambios inesperados y por tanto se han de contemplar los riesgos más comunes y la aparición de imprevistos.
Mientras que considero lógico, plausible y cabal la inclusión de una metodología para la gestión de los riesgos, no comparto completamente la visión de producto finalizado o línea de meta. Establecer una única línea de meta limita el proyecto y además no es realista, es más coherente establecer "mini-metas", sprints, en los cuales se cumplen un conjunto de funcionalidades o requisitos, aunque todos sabemos que esto no es siempre posible.

Para finalizar este punto, me gustaría hacer referencia a una muy buena reflexión que nos muestra Trussell al final de su documento, una metodología de software no es un dogma. Es una frase que debería ser enmarcada y tenerla muy presente a lo largo de cualquier proyecto de software. Un proyecto puede seguir una metodología muy madura, muy completa, ¡genial!, orientada a la calidad, que mantiene unos estándares muy elevados y a la vez que sus usuarios decidan tomar ocasionalmente algunos atajos o modificar alguna de las prácticas. Una metodología no deja de ser una herramienta y debe ser tratada como tal. El juicio de los profesionales de la compañía se debe tener muy en cuenta, ya que a veces nos encontraremos con situaciones donde la metodología debe ser rota en beneficio del proyecto.

¿Es necesaria una metodología de desarrollo?

Es evidente que necesitamos una metodología de trabajo, unas pautas a seguir que nos ayuden a coordinar las complejas tareas que suponen el desarrollo de software.
Pero desde mi punto de vista, la pregunta más interesante sería ¿Cómo debe ser esa metodología? ¿Cómo de rígida debe de ser? ¿Existe una metodología única?
A lo largo de este proyecto intentaremos responder a estas preguntas y mostrar cual es el mejor camino a seguir en cada situación.
Después de leer las breves líneas de esta sección, seguramente se sentirá decepcionado respecto a la respuesta, pero mi intención no era ni mucho menos realizar una disertación sobre le tema, sino hacer que el lector reparase en esta cuestión y de la misma forma que he hecho yo, encuentre su propia respuesta al respecto. ¿Eres un metodista o de que bandos estas?

Metodologías tradicionales

La primera manera de desarrollar software fue un poco anárquica, íbamos abriendo camino, explorando el terreno y descubriendo los pequeños detalles y las principales características de una profesión incipiente. Nuestro segundo intento ya no fue tan inocente, vino con un título importante a sus espaldas, Ingeniería del software, y todos sabemos que un ingeniería es algo serio, que promete y se compromete a realizar el trabajo siguiendo una metodología, una guía de procesos que garantice la realización del producto. Estas primeras metodologías sentaron las bases de lo que hoy conocemos como metodologías tradicionales y a pesar de que puedan haber cambiado de nombre, mejorado sustancialmente e incrementado su grado de adaptación, siempre llevarán el cartel de metodologías heavyweight (pesadas).
Veamos a continuación las principales características que representan a las metodologías tradicionales:

- Requisitos fijados a lo largo de todo el proyecto.
- Basadas en los procesos.
- Proyectos muy bien documentados.
- Gestión predictiva de los proyectos.
- No siguen ni los principios, ni las técnicas de las metodologías ágiles.

El término de metodologías tradicionales prácticamente nació junto con las metodologías ágiles, es debido a esto que podemos englobar prácticamente a todas las metodologías no ágiles como metodologías que siguen un enfoque tradicional. De este modo podemos referirnos a la siguiente sección sobre metodologías ágiles para determinar las características que no tienen las metodologías tradicionales.

Selección de metodologías

Es difícil escoger una representación de las metodologías tradicionales y que quedemos lo suficientemente satisfechos con ella. Esto es debido a la gran variedad a la que nos enfrentamos y a lo dispares que pueden llegar a ser. Todas ellas cumplen las características que hemos enumerado en el apartado anterior, pero como habréis apreciado, no damos demasiada información sobre ellas, sino que tan solo mencionamos aspectos que están muy extendidos y que podríamos llamar "típicos tópicos" de las metodologías tradicionales. La solución que hemos encontrado y que hemos llevado a cabo, es presentar dos metodologías tradicionales, la primera que fue acuñada como tal, Waterfall y una de las más conocidas y utilizadas actualmente, RUP.

Para su caracterización hemos seguido los puntos que se muestran en *Agile Software development methods* [5], para el análisis de metodologías:

- Procesos.
- Roles y responsabilidades.
- Prácticas.
- Adopción y experiencias.
- Entorno de uso.
- Estudios actuales.

Waterfall

W.W Royce publico en 1970[6] un documento en el cual presentaba un modelo secuencial de desarrollo del software (figura 5), indicando sus defectos y los consejos para mejorarlo y evitar sus errores. Daniel y Yeats obviarían los problemas mencionado por Royce y describirían el tradicional modelo de procesos de ciclo de vida de desarrollo de sistemas, comúnmente conocido como waterfall.

Este modelo de procesos define un ciclo de vida que marcó una época en el desarrollo del software y que actualmente vuelve estar en boca de todos, al ser escogido por los practicionistas ágiles para representar las metodologías tradicionales y estar presente en gran cantidad de comparativas que muestran, según mi parecer, unas debilidades excesivamente idóneas para el tema.

1. Proceso

El modelo waterfall se caracteriza por ser un modelo secuencial. Secuencial quiere decir que las etapas se llevan a cabo una detras de la otra, sin superponerse temporalmente. Royce presento las siguientes siete etapas(ver figura 2):

- Captura requisitos del sistema.
- Captura requisitos del software.
- Análisis.
- Diseño del programa.
- Codificación.
- Pruebas.
- Operaciones (Implantación y mantenimiento)

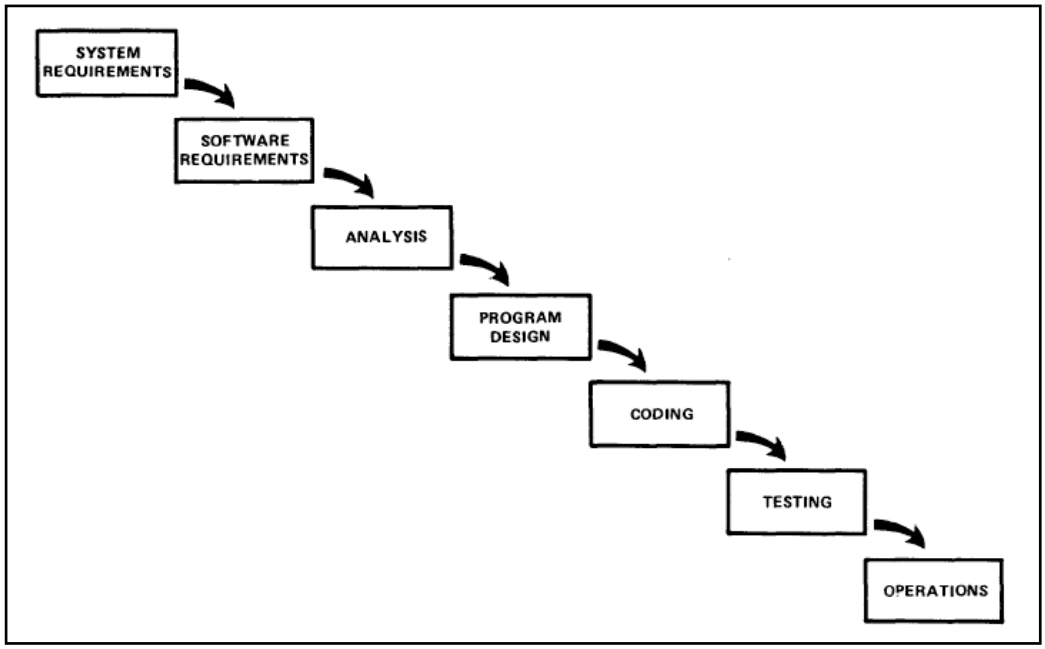


fig 2[4]

Estas siete etapas suelen presentarse en seis o cinco etapas, que deben realizar las siguientes tareas:

- Análisis y especificación de los requisitos.
  Las tres primeras etapas son las encargadas de analizar el problema a resolver y establecer los objetivos de la aplicación a construir, así como sus requisitos. Los requisitos son inamovibles a lo largo del proyecto.

- Diseño.
  Las especificaciones del sistema se trasladan a una representación del software. Un ingeniero

4 La figura que podemos ver arriba esta extraída del paper original[8] donde Royce expuso este modelo de procesos secuencial, que se convertiría en el conocido waterfall.

del software debe preocuparse en esta etapa de:

- Estructura de los datos.
- Arquitectura del software.
- Detalles algorítmicos.
- Representación de la interfaz.

* Codificación.
  El diseño de la aplicación se traslada al dominio de la aplicación o lo que es lo mismo, se realiza mediante una tecnología concreta la solución diseñada.
* Pruebas.
  Su principal cometido es detectar y corregir los errores e identificar que el software cumple la especificación inicial.
* Implantación y mantenimiento.
  A lo largo de esta fase el software debe actualizarse para:

  - Adaptarse a los nuevos requisitos de los clientes.
  - Adaptarse a los diferentes entornos de trabajo.
  - Corregir errores no detectados en la fase de pruebas.
  - Mejorar la eficiencia de la aplicación.

2. Roles y responsabilidades

Podemos identificar los siguiente roles en el desarrollo del software con Waterfall:

* Gestor de Proyectos.
  Es el encargado de gestionar y planificar el proyecto. Esta involucrado en todas y cada una de las etapas del proyecto y es el encargado de controlar que se siguen los planes temporales establecidos inicialmente, así como los requisitos especificados.

* Arquitecto del software.
  Involucrado en las etapas de diseño y desarrollo de la aplicación, es el encargado principalmente de:

  - Diseño de la aplicación.
  - Decisiones estratégicas relevantes para futuras evoluciones tecnológicas del producto o adaptabilidad a entornos diferentes.
  - Diseño del comportamiento de la aplicación con respecto a otros sistemas software.
  - Comunicar su visión del producto adecuadamente para ser realizado según sus indicaciones en la fase de desarrollo. Esta visión debe coincidir con la especificada por los clientes.

* Analistas.
  Son los encargados de estudiar el dominio de la solución y especificar los requisitos de la aplicación. Están involucrados principalmente en las fases de captura y especificación de requisitos, aunque también participan de la fase de diseño como comunicadores de las necesidades del producto.

* Desarrolladores.
  Son los encargados de realizar tecnológicamente la solución, implementarla siguiendo las indicaciones que el arquitecto ha plasmado en el diseño. Están involucrados en las fases de desarrollo, pruebas y mantenimiento.

- Probadores.
  Son los encargados de llevar a cabos las pruebas necesarias para asegurar que la aplicación funciona correctamente y cumple las especificaciones de los clientes.

- Expertos en la materia.
  En cada proyecto es necesario un experto que ayude a entender la naturaleza del problema y la solución requerida. De la misma forma que los analistas participan de las fases de especificación de requisitos y diseño, los expertos también deben estar presentes en estas etapas. Según la wikipedia, un Subject Matter Expert es una persona que es un profesional con experiencia en un campo de la aplicación, pero que no tiene conocimientos técnicos del proyecto.

3. Prácticas

No especifican ninguna práctica concreta.

4. Adopción y experiencias

Es uno de los modelos más utilizadas a lo largo del siglo pasado y que se sigue enseñando como base para los futuros ingenieros del software. Ha sido utilizado en proyectos del departamento de defensa estadounidense, la NASA[7] y multitud de empresas de desarrollo de software.

5. Entornos de uso

La utilización de waterfall debería estar limitada a situaciones en las que los requisitos están muy bien determinados y no existe posible una mala interpretación de los mismos. Estos requisitos deben ser invariables a lo largo del proyecto. El mejor ejemplo de uso correcto de este modelo es en proyectos que a han sido realizados con anterioridad y en los cuales tenemos mucha experiencia, por ejemplo una empresa que desarrolla sistemas de contabilidad y les piden desarrollar un sistema de contabildiad idéntico a otros que hizo con anterioridad, en este caso waterfall es un modelo que funciona muy bien.

6. Estudios actuales

Actualmente es un modelo que esta siendo muy referenciado por las diferentes corrientes ágiles, a la hora de ensalzar las virtudes de la adaptabilidad en comparación con un modelo secuencial. La mayor parte de mejoras y estudios sobre este modelo fueron llevados a cabo el siglo pasado, dando origen a gran cantidad de metodologías basadas en su ciclo de vida. A continuación veremos una metodología que se basa o que tiene sus inicios en el modelo waterfall.

RUP

Rational Unified Proces o RUP, es un framework de procesos de desarrollo de software, ampliamente utilizado en miles de proyectos, con equipos que van de dos a cientos de miembros y utilizado en una amplia variedad de compañías a lo largo del mundo. Fue creado por la Rational Software Corporation. En la figura 3 podemos observar la evolución de la metodología RUP desde sus orígenes. RUP es una metodología basada en Objectory, metodología orientada a objetos creada por Ivar Jacobson[5] en 1988. Rational compró Objectory en 1995 y siete años más tarde IBM se hizo con el control de Rational Corporation.[9]

---

5 Ivar Jacobson, Grady Booch y James Rumbaugh son "the Three amigos" y lideraron la creación del lenguaje unificado de modelado UML.

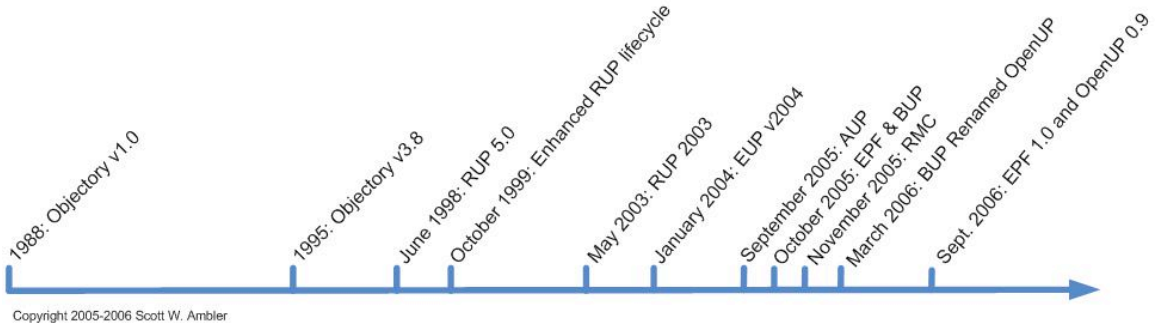


fig 3

RUP no es proceso prescriptivo, sino que es un framework de procesos adaptable y con la intención de ser personalizado por los diferentes equipos de desarrolladores, seleccionando los elementos del proceso que son mas apropiados para sus necesidades. Rational Unified Process es tambien un producto que distribuye IBM y que incluye una gran base documental con artefactos de ejemplo y detalladas descripciones de las diferentes actividades.

1. *Procesos.*

El ciclo de vida de RUP es UP o proceso unificado y describe la dimensión temporal del proyecto, esto es como un proyecto es dividido en fases e iteraciones. UP divide el proyecto en cuatro fases: Inicio, Elaboración, Construcción y Transición[8], las cuales las podemos ver en la figura 4, junto con las disciplinas que se desarrollan en cada una. Veamos con mayor detalles estas fases:

- Inicio.
  En esta fase se establece el objetivo principal del sistema y también se trata de entender que sistema se va a construir, a través de los requisitos de los usuarios. Otras tareas que se realizan en esta fase es la identificación y mitigación de los riesgos de negocio típicos, generación de los casos de negocio para construir el sistema y el documento Vision, con tal de obtener la aceptación de todos los stakeholders.

- Elaboración.
  En la fase de elaboración se busca reducir al máximo los posibles riesgos y de este modo cumplir la planificación y los costes estimados. Los riesgos técnicos son tratados con especial interés, prestando una mayor atención a las tareas técnicamente más difíciles. Aquí se desarrollan las disciplinas de diseñar, implementar, probar y como punto de partida se genera una arquitectura ejecutable que incluye subsistemas, sus interfaces, componentes claves, y detalles de como se comunican los procesos internos o como se realiza la persistencia.

- Construcción.
  Se lleva a cabo la mayor parte de la implementación del sistema, partiendo de una arquitectura ya ejecutable construida en la etapa anterior hasta una primera versión funcional del sistema. A continuación se desplegarán diferentes versiones, internas y alpha[6], con tal de asegurar la usabilidad del sistema y que cumple las necesidades de los usuarios A final de la fase se libera un versión beta completamente funcional, que incluye la documentación necesaria para su instalación, soporte y aprendizaje.

6 Un versión alpha puede ser publica o privada, pero a diferencia de las versiones beta, no están implementadas todas las funcionalidades.

- Transición.
  En esta fase se asegura que el software cumple las necesidades de los usuarios y esto se hace a través de la realización de pruebas y la realización de pequeños ajustes basados en el feedback con el cliente. En esta fase no se deberían encontrar problemas estructurales mayores, ya que deberían haber sido identificados y corregidos en etapas anteriores.

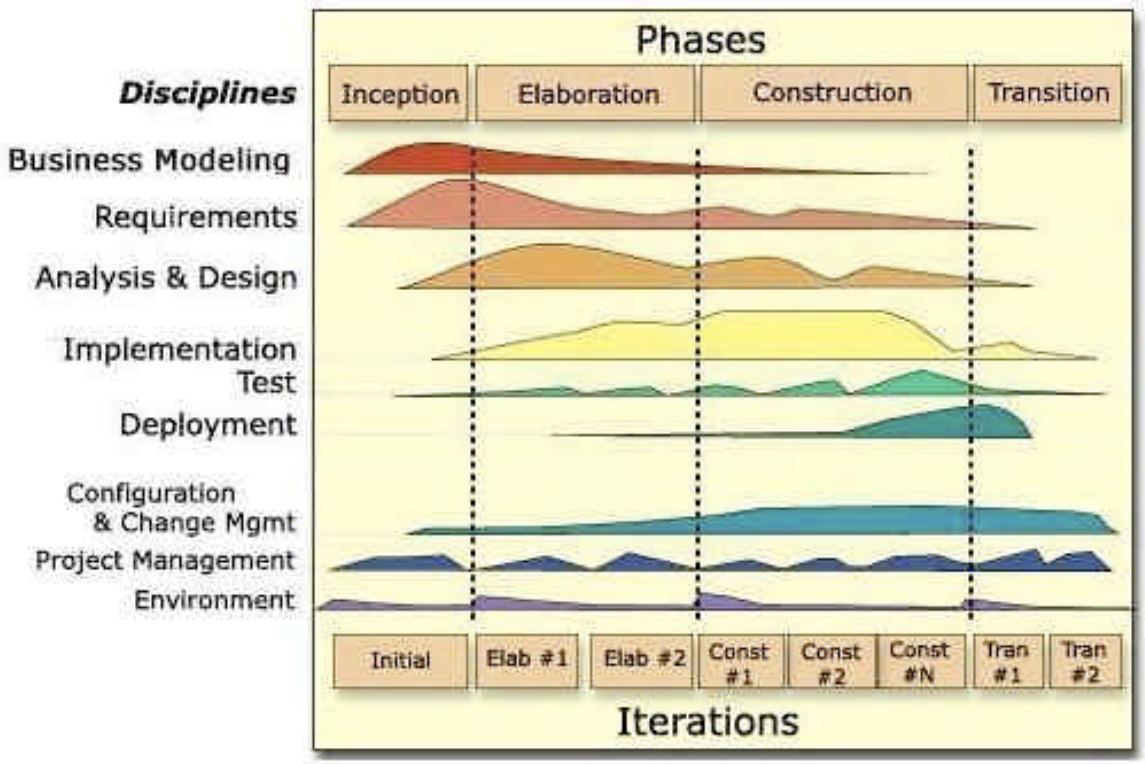


fig 4. The RUP v2003 lifecycle

Cada una de las fases contienen una o mas interacciones, que se centran en producir un incremento del producto, el cual puede ser código u otros entregables necesarios para alcanzar los objetivos de negocio de la fase. Para ver información sobre los procesos o disciplinas que se desarrollan en RUP les sugerimos las siguientes referencias [8][11].

2. *Roles y responsabilidades.*

Anthony Crain publicó un artículo[10] en el cual se muestran los diferentes roles que podemos encontrar en un proyecto RUP, de una forma muy intuitiva y de las mejores que he visto al respecto. En la siguiente tabla [tabla 1] observamos los roles según su grado de detalle, tal y como se trabaja con RUP, en primera instancia se obtiene un visión general de la solución y después, en cada nueva iteración se concretan los detalles. De esta forma, especificamos los roles según si pertenecen a una de las primeras iteraciones (roles generales) o a iteraciones más avanzadas (roles específicos).

Disciplina	Roles generales	Roles específicos
Modelado de negocio	Analista de procesos de negocio. Descubrir todos los casos de uso de negocio.	Diseñador de negocio. Detallar un conjunto de los casos de uso de negocio.
Requisitos	Analista de sistemas. Descubrir todos los casos de uso.	Especificador de casos de uso. Detallar un conjunto de los casos de uso.
Análisis y diseño	Arquitectos de Software. Toma decisiones tecnológicas de la solución a nivel global.	Diseñadores. Detallan el análisis y diseño para un conjunto de casos de uso.
Implementación	Integrador. Es el propietario del plan de construcción que muestra como se integrarán cada una de las clases, las unas con las otras.	Desarrollador o programador. Implementa un conjunto de clases o un conjunto de operaciones de una clase.
Pruebas	Gestor de las pruebas. Asegura que las pruebas han sido realizadas correctamente. Analista de pruebas Selecciona que se va a probar según lo estimado. Diseñador de pruebas Decide que pruebas deberían ser automáticas o manuales, y crea las automáticas.	Diseñador de pruebas. Implementa las pruebas automáticas de la iteración. Probador. Ejecuta un test específico.
Despliegue o Implantación	Gestor de la implantación. Supervisa la implantación de todas las unidades.	Artista gráfico, escritor tecnológico y desarrollador de material. Crean el material necesario para asegurar la correcta implantación.
Gestión del proyecto	Gestor del proyecto. Crea los casos de negocio y un plan general, y toma decisiones críticas al respecto de que cosas hacer y cuales no hacer.	Gestor de proyectos. Planifica, monitoriza y gestiona los riesgos para una sola iteración.
Entorno	Ingeniero de procesos. Es el responsable de los procesos del proyecto.	Especialista en herramientas. Crea manuales de uso de herramientas específicas.

Disciplina	Roles generales	Roles específicos
Configuración y Mantenimiento	Gestor de la configuración. Establece las políticas y planes. Gestor de control de cambios. Establece un proceso de control de los cambios.	Gestor de la configuración. Crea una unidad de despliegue o implantación, reportes del estado de la configuración, auditorias, ... Gestor de control de cambios. Revisa y gestiona las peticiones de cambios.

tabla 1

3. *Prácticas.*

RUP presenta un gran numero de prácticas tal y como podemos ver en *Agility and Discipline*[8] o en "RUP, Best software practices development"[11], a continuación mostramos en la figura 9 las prácticas que hemos considerado más relevantes.

Gestión de los riesgos	Decidir que riesgos tener en cuenta en cada iteración. Actualizar la lista de riesgos y hacerla visible. Utilizar herramientas de gestión de riesgos y trazabilidad.
Desarrollos iterativos	Entregas iterativas. Replanificación basada en el feedback. Planificar las iteraciones en función de los riesgos. Usar mini y super iteraciones.
Usar componentes basados en la arquitectura	Aplicar componentes y principio de diseño de servicios. Modelar componentes, servicios e interfaces. Crear especificaciones detalladas de los componentes y los servicios.
Gestión de los requisitos	Identificar escenarios. Capturar casos de uso, y relacionarlos con sus respectivos escenarios. Trocear casos de uso en requisitos gestionados independientemente.
Modelado del software visual	Utiliza UML para la realización de un modelo visual del software.
Verificar la calidad del software	La calidad debe ser revisada respecto a los requisitos basándose en la fiabilidad, funcionalidad, rendimiento de la aplicación y del sistema. Planificación, diseño, implementación, ejecución y evaluación de las pruebas.
Control de los cambios del software	Controlar y hacer el seguimiento de los cambios. Utilización de entornos de trabajos independientes para aislar los diferentes cambios.

tabla 2

4. *Adopción y experiencias.*

RUP es una de los modelos de proceso más utilizados hoy en día y prácticamente se ha convertido en un estándar para procesos prescriptivos de desarrollo de software. Si lo que queremos es ver la cantidad de empresas que han utilizado o que están utilizando RUP, siempre podemos dirigirnos a la página oficial de IBM y observar los casos de estudio que nos muestran[12]. Entre estos casos de estudio podemos encontrar las divisiones médicas y de semiconductores de la multinacional Philips, el portal de liga de fútbol americano de EEUU (NFL) o a la consultora Morris.

5. *Entornos de uso.*

IBM provee de una herramienta llamada RMC o IBM Rational Method Composer[8] que propone los siguientes procesos en los cuales se pueden ejecutar RUP:

- RUP para proyectos pequeños.
- RUP para proyectos de mediana envergadura.
- RUP para proyectos grandes. Y especifica que es el RUP clásico.
- RUP para COTS o desarrollo de aplicaciones empaquetadas.
- RUP para ingeniería de sistemas.
- RUP para arquitecturas orientadas a servicios (SOA).
- RUP para mantenimiento.

Podemos observar que desde IBM prácticamente recomiendan el uso de RUP para cualquier tipo de proyecto, grandes o pequeños, proyectos para COTS o de mantenimiento, etc. Obviamente esta es la sugerencia de IBM y al menos nos indica que RUP considera su utilización en todos estos ámbitos, otro tema es que sea la metodología más idónea en cada caso. Según mi parecer RUP puede ser muy útil para grandes proyectos y de mediana envergadura, pero cuando hablamos de proyectos pequeños, añade un trabajo excesivo que podría ser fácilmente obviado.

6. *Estudios actuales.*

Durante la última década, RUP ha continuado evolucionando, adoptando nuevas prácticas y lo ha hecho a través de la integración de otros procesos, como el Objectory Process, SUMMIT Ascendant, la colaboración con importantes compañías e instituciones, como SEI Carnegie Mellon, USC Center for Computer Engineering ... y de los esfuerzos de la gran comunidad de desarrolladores de software con sus sugerencias y experiencias personales.

Metodologías ágiles

En el segundo apartado de este capítulo, *Contexto histórico,* hemos presentado un recorrido cronológico de las metodologías, clasificándolas según su enfoque y comentando sus características más esenciales. Al final de este recorrido nos encontramos con las metodologías actuales, que se fueron gestando a finales del siglo pasado y que han eclosionado a principios del actual. Tal y como concluíamos, las metodologías ágiles surgen como una alternativa, una reacción a las metodologías tradicionales y principalmente a su burocracia. Muchas ideas que se plantean en las metodologías ágiles no son nuevas, gran parte de ellas ya fueron reflejadas por Brooks en su mítico libro, The Mythical Man Month[2] y en gran parte responden al sentido común. Algunos autores consideran que se ha cumplido un círculo que empezó con una reacción provocada por múltiples factores y señalada temporalmente por el manifiesto de Dijkstra, en el cual se hacía un llamamiento a la disciplina y que se cierra con el ya famoso Manifest for Agile Software Development, una petición por la relajación de los procesos en pro de las personas.

La aparición de las metodologías ágiles no puede ser asociada a una única causa, sino a todo un conjunto, bien es cierto que la mayoría de autores nos indicarán que son una reacción a las metodologías tradicionales, pero ¿cuales fueron las causas de esta reacción?, me gustaría indicar los factores que comúnmente más se mencionan y los que yo realmente consideró que hicieron realidad esta nueva manera de desarrollar software:

- "Plumbing". La traducción al castellano sería pesadez, lentitud de reacción, exceso de documentación, en definitiva, falta de agilidad de los modelos de desarrollo existente.
- Las metodologías existentes no cumplieron las expectativas planteadas inicialmente.
- Explosión de la red y las aplicaciones Web.
- Movimiento open source.

Podríamos indicar algunas causas más, pero creo que en definitiva todas ellas tendrían origen en estas cuatro. Definitivamente, las metodologías tradicionales exigen un sobreesfuerzo por parte del equipo de desarrollo en fases poco productivas, como es la documentación y debido a su estructuración (típicamente siguiendo el modelo waterfall o más actualmente UP) son poco flexibles a los cambios, de requisitos, de nuevas funcionalidades, etc. Si a todo esto le añadimos que las metodologías tradicionales no han sido capaces de eliminar todos los fallos, que persiguen al desarrollo de proyectos software prácticamente desde sus inicios, obtenemos un clima un poco escéptico respecto a las metodologías. Ahora a este mejunje le vamos a añadir un cambio bastante importante, en cuanto a la demanda del mercado del software, cada vez más orientada a la Web, con uno requisitos muy volátiles, que requieren tiempos de desarrollo cada vez más cortos y con una comunidad "in crescendo", la comunidad del software libre. El éxito creciente de productos open source y de las crecientes comunidades de desarrolladores open source, provocó unas reacciones muy interesantes, las empresas se fijaban en nuevos desarrolladores, con nuevos métodos y ¡que funcionaban!, se produce una simbiosis y parece que poco a poco se utilizan métodos "amateurs" que se combinan a las metodologías formales. Los modelos de desarrollo de las comunidades open source pudieron ciertamente determinar la aparición de las metodologías ágiles, pero supongo que según a quien preguntes, te indicará un factor más determinante que otro, es por lo que creo que fue una combinación de todos ellos que dieron lugar a un modo de desarrollo del software más ágil, que no tiene porque ser mejor, aunque realmente lo parece.

¿Qué significa ser ágil?

El objetivo de los métodos ágiles es permitir que una empresa sea ágil, pero, ¿qué significa ser ágil? Jim Highsmith dice que ser Agile significa ser capaz de "Deliver quickly, Change quickly, Change often"[7] . Mientras que las técnicas ágiles pueden variar en su ejecución y en su énfasis, todas ellas comparten características, incluyendo el desarrollo iterativo y que están centradas en la interacción, comunicación, y en la reducción de la creación de artefactos intermedios. Desarrollar mediante iteraciones permite al equipo de desarrollo adaptarse a los cambios de los requisitos. Trabajar con un alto grado de comunicación permite que el equipo pueda tomar decisiones y aplicarlas inmediatamente. La reducción de artefactos intermedios que no dan valor añadido al producto final, nos permite asignar más recursos al producto en si y podrá ser acabado antes.
Cockburn y Highsmith nos explican que lo que es realmente nuevo de los métodos ágiles, no son las prácticas que ellos utilizan, sino el reconocimiento de las personas como principales valedoras para que un proyecto triunfe, en conjunto con una gran orientación a la efectividad y la manejabilidad. La mayoría de seguidores de las metodologías ágiles están de acuerdo en que ser ágil, es algo mas que seguir las guías que se supone que hacen un proyecto ágil. La verdadera agilidad es más que un conjunto de prácticas, es un estado de ánimo. Andrea Branca va más allá y afirma que "otros procesos pueden parecer ágiles, pero ellos no se sienten ágiles". Estas referencias son las que nos hacen entender mejor que se use la terminología de religiones, cuando hablamos de metodologías, ya que muchas veces se aducen a factores que trascienden de lo meramente formal y racional, para pasar al terreno de las creencias y sentimientos. Yo en concreto percibo con mayor "feeling" las metodologías ágiles, debido a su orientación social, pero intento no olvidar mi estatus de ingeniero y todo lo que ello acarrea. De esta forma, el punto vista de las metodologías ágiles que prefiero utilizar esta más próximo al de esta cita:

"*El factor más importante en el desarrollo de software no son las técnicas y las herramientas que emplean los programadores, sino la calidad de los propios programadores*"

Robert L. Glass. Facts and Fallacies of Software Engineering

La calidad de los programadores determinará en un grado muy elevado el éxito del proyecto, parece una frase muy tonta y evidente, pero por alguna extraña razón parece que no todo el mundo la tiene presente.

Los cuatro principios del manifiesto ágil

En marzo de 2001, Kent Beck convocó a 17 críticos de los modelos de mejora basados en procesos, justo un par de años antes Kent había publicado el libro "Extreme Programming Explained" en el que exponía una nueva metodología denominada Extreme Programming, se reunieron en Salt Lake City para discutir sobre el desarrollo de software.
En la reunión se acuñó el término "Métodos Ágiles" para definir los métodos que estaban surgiendo como alternativa a las metodologías formales, consideradas excesivamente "pesadas" y rígidas por su carácter normativo y su fuerte dependencia de planificaciones detalladas, previas al desarrollo. De esta reunión surgiría un resumen en forma de cuatro principios, que actualmente conocemos como "Manifiesto Ágil"[tabla 3] , que básicamente son los principios sobre los que se basan los métodos reconocidos como ágiles. Hasta 2005, entre los defensores de los modelos de procesos y los de modelos ágiles han sido frecuentes las posturas radicales, quizá más ocupadas en descalificar al otro que en estudiar sus métodos y conocerlos para mejorar los propios.

7 Entregar rápido, cambiar rápido, cambiar con frecuencia.

Manifiesto Ágil (http://www.agilemanifesto.org)
Estamos poniendo al descubierto mejores métodos para desarrollar software, haciéndolo y ayudando a otros a que lo hagan. Con este trabajo hemos llegado a valorar: ✱ A los individuos y su interacción, por encima de los procesos y las herramientas. ✱ El software que funciona, por encima de la documentación exhaustiva. ✱ La colaboración con el cliente, por encima de la negociación contractual. ✱ La respuesta al cambio, por encima del seguimiento de un plan.
Firmado por: Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas.

tabla 3

1. *Valoramos más a los individuos y su interacción que a los procesos y las herramientas.*

Se puede considerar el principio más relevante de todo el manifiesto. No desvirtuamos, ni quitamos importancia a los procesos, que evidentemente son importantes, ayudan al trabajo y son una guía de operación. Las herramientas mejoran la eficiencia, pero en trabajos que requieren conocimiento implícito, sin personas con conocimiento técnico y actitud adecuada, no producen resultados. A modo de pancarta publicitaria, las empresas suelen predicar muy alto que sus empleados son lo más importante, pero la realidad es que en los años 90 la teoría de producción basada en procesos, la re-ingeniería de procesos ha dado a éstos más relevancia de la que pueden tener en tareas que deben gran parte de su valor al conocimiento y al talento de las personas que las realizan. Los procesos deben ser una ayuda y un soporte para guiar el trabajo. Deben adaptarse a la organización, a los equipos y a las personas; y no al revés. La defensa a ultranza de los procesos lleva a postular que con ellos se pueden conseguir resultados extraordinarios con personas mediocres, y lo cierto es que este principio es peligroso cuando los trabajos necesitan creatividad e innovación.

2. *Valoramos más el software que funciona que la documentación exhaustiva.*

Poder ver anticipadamente como se comportan las funcionalidades que se esperan sobre prototipos o sobre partes ya elaboradas del sistema final ofrece un "feedback" muy estimulante y enriquecedor que genera ideas y posibilidades imposibles de concebir en un primer momento, y difícilmente se podrían incluir al redactar un documento de requisitos detallados antes de comenzar el proyecto. El manifiesto no afirma que no hagan falta. Los documentos son soporte de documentación, permiten la transferencia del conocimiento, registran información histórica, y en muchas cuestiones legales o normativas son obligatorios, pero se resalta que son menos importantes que los productos que funcionan. Menos trascendentales para aportar valor al producto. Los documentos no pueden sustituir, ni pueden ofrecer la riqueza y generación de valor que se logra con la comunicación directa entre las personas y a través de la interacción con los prototipos. Por eso, siempre que sea posible debe preferirse, y reducir al mínimo indispensable el uso de documentación, que genera trabajo que no aporta un valor directo al producto. Si la organización y los equipos se comunican a través de documentos, además de perder la riqueza que da la interacción con el producto, se acaba derivando a emplear a los documentos como barricadas entre departamentos o entre personas.

3. *Valoramos más la colaboración con el cliente que la negociación contractual.*

Las prácticas ágiles están especialmente indicadas para productos difíciles de definir con detalle en un principio, o que si se definieran así tendrían al final menos valor que si se van enriqueciendo con retro-información continua durante el desarrollo. También para los casos en los que los requisitos van a ser muy inestables por la velocidad del entorno de negocio. Para el desarrollo ágil el valor del resultado no es consecuencia de haber controlado una ejecución conforme a procesos, sino de haber sido implementado directamente sobre el producto. Un contrato no aporta valor al producto. Es una formalidad que establece líneas divisorias entre responsabilidades, que fija los referentes para posibles disputas contractuales entre cliente y proveedor. En el desarrollo ágil el cliente es un miembro más del equipo, que se integra y colabora en el grupo de trabajo. Los modelos de contrato por obra no encajan.

4. *Valoramos más la respuesta al cambio que el seguimiento de un plan*

Para un modelo de desarrollo que surge de entornos inestables, que tienen como factor inherente el cambio y la evolución rápida y continua, resulta mucho más valiosa la capacidad de respuesta que la de seguimiento y aseguramiento de planes pre-establecidos. Los principales valores de la gestión ágil son la anticipación y la adaptación; diferentes a los de la gestión de proyectos ortodoxa: planificación y control para evitar desviaciones sobre el plan.

Criterios selección metodologías

A la hora de seleccionar las metodologías tradicionales, no hemos realizado ningún estudio previo, ni seguido unos criterios muy estrictos, simplemente hemos seleccionado dos metodologías que son fácilmente reconocibles por cualquier ingeniero del software y profesionales del sector y que nos permitían sacar a relucir sus características principales. Podríamos haber hecho lo mismo en el caso de las metodologías ágiles, pero debido a que son el eje central del documento y que el estudio posterior será más exhaustivo, queríamos garantizar tener la suficiente información de las metodologías con las que íbamos a trabajar y que fuesen lo suficientemente relevantes.
El objetivo de este apartado es seleccionar un total de 5 metodologías ágiles, que posteriormente serán caracterizadas y sobre las cuales realizaremos una comparativa en el capítulo siete. Las metodologías que hemos considerado a la hora de realizar la selección han sido:

- Adaptative Software Development
- Agile Modeling
- Agile Model Driven Development
- Agile Project Management
- Agile Unified Process
- Crystal Methods
- Dynamic Systems development methods
- Feature driven development
- Internet Speed Development
- Lean development
- Pragmatic programming
- Scrum
- Test Driven Development
- XBreed
- Extreme Programming
- Win Win Spiral.

Todas estas metodologías están documentadas o referenciadas como metodologías ágiles en las siguientes referencias [1][13][14][15][16][17]

No son todas las metodologías ágiles que existen, pero si que son una gran representación de ellas. A continuación mostramos algunas metodologías ágiles que no hemos incluido en la preselección:

- Evolutionary Project Management.
- Story cards driven development.
- Agile Unified Process
- Open Unified Process

Tal y como hemos mencionado un poco mas arriba, nos interesa trabajar con metodologías lo suficientemente documentadas, que nos faciliten la obtención de información, pero también es interesante trabajar con metodologías que dispongan de algún tipo de certificación y training. Según estas condiciones hemos determinado seis clasificaciones donde escogemos a las cinco metodologías que se encuentran mejor posicionadas, estas son las clasificaciones:

1. La metodología con mayor presencia en Internet.
2. La metodología mejor documentada.
3. Metodologías certificadas y con training.
4. Metodologías con comunidades.
5. Metodología más utilizada por empresas. Presencia empresarial.
6. Metodología más utilizada en proyectos software.

En la tabla 4 mostramos algunas consideraciones que hemos tenido a la hora de realizar estas clasificaciones y en la tabla 5, los resultados de los datos que hemos considerado para elaborar las listas.

Libros	Safari Books, Google Books, Amazon, Abacus, Díaz de Santos, CCUC
Certificación y Training	Se han considerado metodologías certificadas aquellas que emiten un certificado que aseguran el cumplimiento y seguimiento de la metodología, así como sus técnicas y prácticas. Indicamos que una metodología dispone de training, si hemos encontrado alguna institución, organización o compañía que ofrezca formación de la metodología. *XP esta especialmente bien documentada con innumerables recursos on-line disponibles, comunidades libres y grupos de noticias.
Comunidades	Contemplamos tanto si ha formado una comunidad relevante o si esta asociada a la Agile Alliance, soportándola y cumpliendo sus principios. No se incluyen grupos de noticias o de correo, aunque seria una propuesta interesante a tener en cuenta.
Proyectos realizados	La mayoría de metodologías se han aplicado en empresas privadas y por lo tanto no existe mucha documentación al respecto, no se ha realizado una búsqueda exhaustiva, ya que no es el propósito de este proyecto. * Fuentes en la que podemos encontrar gran cantidad de proyectos realizados con XP: http://www.c2.com/cgi/wiki?ExtremeProgrammingProjects http://www.c2.com/cgi/wiki?SuccessfulXpProjects

tabla 4

Metodologías	NºPapers	Google	Yahoo	Live	Libros en español	Libros en otros idiomas	Certificación/ Training	Comunidades (Alliances)	Presencia empresarial	Proyectos realizados
Adaptative Software Development (ASD)	2	15300	46100	23100	0	1	No	Agile alliance	-	-
Agile Modeling (AM)	6	57200	203000	538000	0	1	Training	Agile alliance	-	-
Agile Model Driven Development (AMDD)	2	10200	28400	83000	0	1	Training	Agile alliance	-	-
Agile Project Management (APM)	8	170000	766000	311000	0	1	Training	Declaration of Interdependence for modern management	-	-
Agile Unified Process (AUP)	0	11000	19700	8940	0	0	No	-	-	-
Crystal Methods	0	244000	2930000	724000	0	1	Training	Agile alliance	-	Proyecto Winifred
Dynamic Systems Development Method (DSDM)	70	16900	41200	24200	0	6	DSDM Certified Training	DSDM Consotium	-	-
Feature Driven Development (FDD)	3	31200	177000	68000	0	1	FDD Certified Training	Agile alliance	-	-
Internet Speed Development (ISD)	0	74	326	127	0	0	No	Agile alliance	-	-
Lean Development	2	16300	6890	16100	0	0	Training	Agile alliance	-	-
Pragmatic Programming	0	346	1340	648	0	0	No	Agile alliance	-	-
Scrum	43	3420000	5120000	1970000	2	4	Scrum Certified Trainning	Agile alliance	Yahoo, Google, etc	Desarrollos internos principalmente
Test Driven Development	55	492000	2800000	1040000	0	9	No	Scrum Alliance Agile Alliance	-	-
XBreed (Agile Enterprise)	0	601	1450	624	0	0	No	Agile Alliance	-	-
Extreme Programming (XP)	+100	1190000	4470000	1470000	2	+20	Training*	-	Chrysler, Sabre Airlines, CSEE Transport, etc....	Control automatizado de trenes *
Win Win Spiral	3	1700	1640	447	0	0	No	Agile Alliance	-	-

tabla 5

Resultados del estudio

En la tabla 5 que mostramos en la página anterior podemos ver los resultados de los criterios escogidos. Estos son los resultados que hemos extraído del estudio previo, según los criterios establecidos anteriormente. Si el lector desea obtener más información relevante respecto a la metodología seguida en la clasificación de las metodologías, criterios de selección considerados, tendencias de búsquedas por Internet y los resultados detallados, le indicamos que se dirija al anexo número uno.

1. *La metodología con mayor presencia en Internet.*

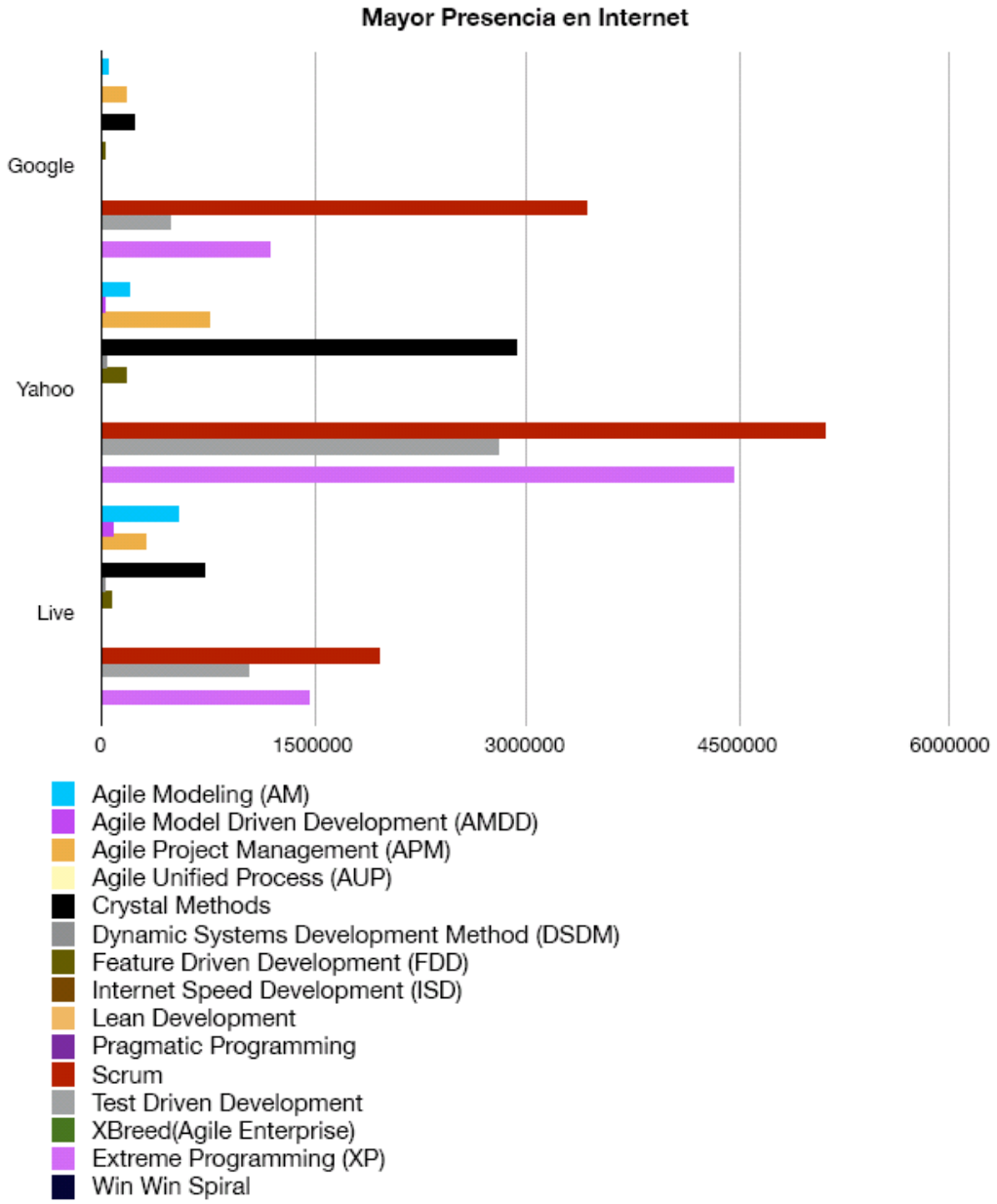


fig 4

Según el número de resultados obtenidos en las búsquedas por Yahoo, Google y Microsoft Live, las 5 metodologías con mayor presencia en la red y en este orden son (ver figura 4):

1. Scrum
2. Extreme Programming (XP)
3. Test Driven Development
4. Crystal Methods
5. Agile Project Management (APM)

2. *La metodología mejor documentada.*

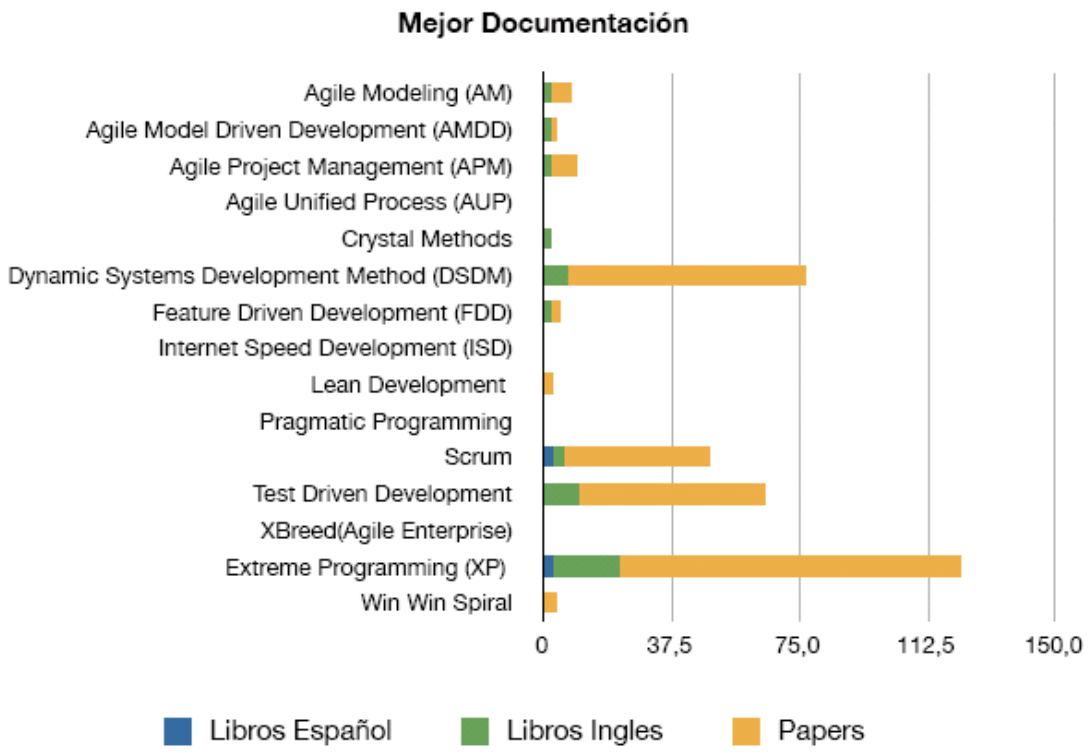


fig 5

Las cinco metodologías mejor documentadas son (ver figura 5):

1. Extreme Programming
2. Test Driven Development
3. Dynamic System Development Method
4. Scrum
5. Agile Project Management

3. *Metodologías certificadas y con training.*

Metodologías con certificación:

- Dynamic System Development Method tiene un DSDM certified.
- Feature Driven Development tiene FDD Certified.
- Scrum dispone de Scrum Certified.

Metodologías con training:

- Agile Modeling.
- Agile Model Driven Development.
- Agile Project Management.
- Crystal methods.
- Dynamic System Development Method.
- Feature Driven Development.
- Lean Development.
- Scrum.
- Extreme Programing.

Con certificación y training:

- Todas las metodologías certificadas ofrecen training.

4. Metodologías con comunidades.

La mayoría pertenecen a la Agile Alliance, pero algunas han montado autenticas comunidades y alianzas a su alrededor.

Metodologías asociadas a la Agile Alliance:

- Agile Modeling.
- Agile Model Driven Development.
- Crystal methods.
- Dynamic System Development Method.
- Feature Driven Development.
- Internet Speed Development.
- Lean Development.
- Pragmatic Programming.
- Scrum.
- Test Driven Development.
- Extreme Programing.
- Win Win Spiral.

Metodologías con comunidades o alianzas diferentes:

- Agile Project Management, con Declaration of Interdependence for modern management.
- Dynamic System Development Method, con DSDM Consortium.
- Scrum, con Scrum alliance.

5. Metodología más utilizada por empresas. Presencia empresarial.

Tal y como habréis podido observar es realmente complicado encontrar ejemplos de proyectos realizados en una empresa privada y con una metodología en concreto. Por lo que los resultados obtenidos en este apartado no se tienen en cuenta en la selección inicial de las metodologías.

6. Metodología más utilizada en proyectos software.

Exactamente igual que el punto anterior.

Selección metodologías para el estudio

El objetivo inicial era seleccionar un máximo de 5 metodologías, pero en vista de los resultados estas han sido las metodologías seleccionadas:

1. Agile Project Management.
2. Crystal Methods.
3. Dynamic Sytems Development Method.
4. Scrum.
5. Test Driven Development.
6. Extreme Programming.

La elección de estas metodologías para la comparativa se ha realizado a partir de los datos presentados en los apartados anteriores. Se han incluido las 5 metodologías mejor documentadas y las 5 con mayor presencia en Internet. Nos encontramos con que son 6 las metodologías que surgen de esta unión, pero debemos tener en cuenta que Test Driven Development es considerado por muchos como una práctica o técnica incluida en Extreme Programming y por tanto que gran parte de su estudio se verá reflejada en esa metodología. Es por esta razón que se ha incluido una sexta metodología dentro del estudio.

Para su caracterización hemos seguido los puntos que se muestran en *Agile Software development methods* [5], de la misma forma que hemos hecho con las metodologías tradicionales:

- Procesos.
- Roles y responsabilidades.
- Prácticas.
- Adopción y experiencias.
- Entorno de uso.
- Estudios actuales.

La mayor parte de las figuras y contenidos que mostramos a continuación han sido extraídos o bien de las respectivas referencias indicadas o en su defecto, si no lo indicamos del estudio llevado a cabo por Pekka Abrahamsson, Outi salo & Jussi Ronkainen, en *Agile Software Development Methods* [5].

Agile Project Management.

Jim Highsmith encabezó una corriente de pensamiento dirigida a modificar los principios de gestión de proyectos , que hasta entonces eran válidos. La aparición de las metodologías ágiles generaron nuevos entornos de desarrollo y nuevas posibilidades que la gestión tradicional de proyectos no se había encontrado antes. Es por esto que Jim propuso en su libro *Agile Project Management* [19], una nueva perspectiva de gestión de proyectos ágiles, es lo que conocemos ahora como gestión de proyectos ágiles y adaptativos. Un año después se firmo un manifiesto llamado declaración de interdependecia que promulgaba los siguientes principios:

Declaración de Interdependecia (http://pmdoi.org/)
Enfoques ágiles y adaptativos que enlazan a las personas, proyectos y valores. Nosotros somos una comunidad de directores de proyectos que tenemos un alto índice de éxito en nuestros proyectos. Para conseguir estos resultados: ✱ Incrementamos el retorno de la inversión centrándonos en la realización continuada de diferentes flujos de valor. ✱ Entregamos resultados fiables, gracias a que nuestro clientes interactuan frecuentemente con nosotros y compartimos responsabilidades del proyecto. ✱ Aceptamos la incertidumbre y la gestionamos a través de iteraciones, anticipación y adaptación. ✱ Animamos el desarrollo creativo e innovador reconociendo el valor de las individualidades y creando un entorno donde puedan marcar la diferencia. ✱ Incrementamos el rendimiento obteniendo mejores resultados, gracias a que delegamos responsabilidades a los equipos y mejoramos la efectividad del equipo distribuyendo estas responsabilidad dentro del mismo equipo.
Firmado por: David Anderson, Sanjiv Augustine, Christopher Avery, Alistair Cockburn, Mike Cohn, Doug DeCarlo, Donna Fitzgerald, Jim Highsmith, Ole Jepsen, Lowell Lindstrom, Todd Little, Kent McDonald, Pollyanna Pixton, Preston Smith and Robert Wysocki. [2005]

tabla 6

El título "Declaración de interdependecia" se puede interepretar de diferentes maneras. Como los miembros que integran un equipo de un proyecto particular, son parte de un todo interdependiente y no de un grupo de individualidades independientes y aisladas. O como los equipos de los proyectos, los stakeholders, los clientes son tambien interdependientes. Los equipos que no reconocen esta interedependebilidad raramente tienen éxito.
Agile Project Management es una metodología ya que promulga unos principios, procesos, prácticas y como no, una filosofía.

1. *Procesos.*

A pesar de que las metodologías ágiles procesan su preferencias por dar un mayor valor a las personas, en vez de a los procesos, considerando a estos estáticos, prescriptivos y difíciles de cambiar, los procesos no son malos per se. Jim en *Agile Project Management* [19] comenta precisamente esto, los procesos deben ceñirse a los objetivos de negocio, si los objetivos de negocio son repetibles, predecibles, entonces un proceso prescriptivo es lo más adecuado, pero si los objetivos de negocio son innovadores, entonces el framework de procesos debe ser ágil, flexible y fácil de adaptar.
Un framework de procesos ágiles, como el que presenta APM[8], necesita:

- ✱ Soportar una previsión, exploración, adaptación a la cultura.
- ✱ Soportar la auto-organización, auto-disclipina de los equipos.
- ✱ Promover la fiabilidad y la consistencia en la medida de lo posible dado el nivel de incertidumbre del proyecto.
- ✱ Ser flexible y fácilmente adaptable.
- ✱ Soportar visibilidad en los procesos.
- ✱ Tener en cuenta el aprendizaje.
- ✱ Incorporar prácticas que soportan cada fase.
- ✱ Proveer de "checkpoints" a la gestión para las revisiones.

8 Siglas de Agile Project Management

La estructura del modelo de APM esta basada en el modelo descrito en *Adaptive Software Develomet (Highsmith 2000)*. APM se compone de las fases Previsión, Especulación, Exploración, Adaptación y Cierre (ver figura 6), dos fases más que en ASD[9], que se compone de Especulación, Colaboración y Aprendizaje.

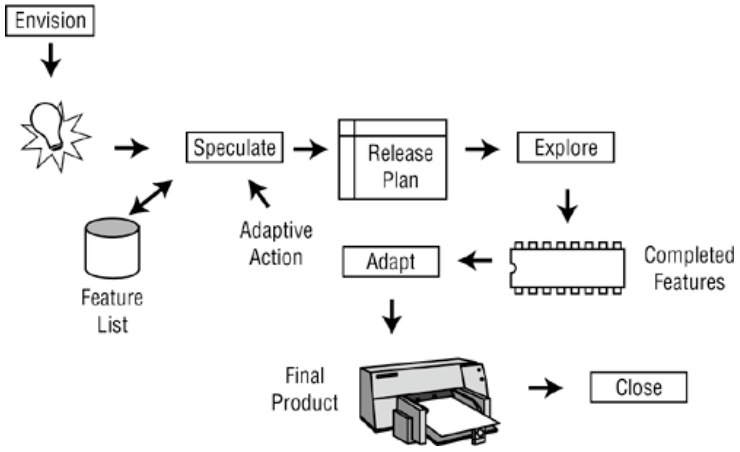


fig 6

Los nombres de cada una de las fases han sido escogidos intencionadamente y con mucho significado, reflejando tanto las actividades como los resultados que se llevan en cada una de ellas. Veamos las cinco fases de APM y en que consiste cada una:

- Previsión.
  En esta fase se determinan la visión del producto, los objetivos del proyecto, la comunidad del proyecto y como el equipo trabajará junto.
  Esta fase sustituye a la tradicionalmente conocida como Inicio y su nombre fue escogido para remarcar la importancia del concepto de visión en esta fase.

- Especulación.
  En esta fase se genera un plan de entregas basado en las funcionalidades del producto más que en las actividades. El plan de entregas utiliza información sobre la especificación del producto, la plataforma de la arquitectura, recursos, análisis de riesgos, niveles de defectos, restricciones de negocio y fechas objetivo.
  Esta fase reemplaza a la tradicionalmente conocida como planificación, su cambio de nombre es debido a las connotaciones que lleva consigo el término planificar. Planificar lleva implícito el concepto de predecir y una relativa certeza respecto a lo que sucederá en un futura, mientras que especular expresa claramente que el futuro es incierto. Debemos aprender a especular y adaptarnos, en vez de planificar y construir.

- Exploración.
  El objetivo de la fase de exploración es obtener funcionalidades de la aplicación probadas y aceptadas. En vez de concentrarse en los detalles técnicos sobre como conseguir este objetivo, un manager de proyectos ágiles se debe centrar en crear equipos auto-organizados, auto-disciplinados que puedan alcanzar los objetivos.
  En este caso, la fase de exploración sustituye a la fase o tarea de gestión del proyecto. La

9 Siglas de Adaptive Software Development

exploración con su estilo iterativo de entregas, es explícitamente no-linear, concurrente y sin seguir el modelo en cascada. Los puntos que se desarrollan en la fase de especulación son explorados, la especulación implica una gran flexibilidad, ya que nos basamos en el hecho de que no podemos predecir los resultados. El modelo de APM da una mayor importancia a la ejecución y al hecho de que es exploratorio, en vez de determinístico.

- Adaptación.
  En la fase de adaptación se revisan los resultados liberados, la situación actual y el rendimiento de la aplicación, y se adapta si es necesario.
  Un equipo que practica APM mantiene su vista siempre en la visión, monitoriza la información, y se adapta a la situación actual.

- Cierre.
  Se concluye el proyecto, se aprende de la experiencia y se celebra. En esta fase lo más importante es la transferencia de conocimientos y según Jim, celebrarlo.

2. Roles y responsabilidades.

Jim concibe una definición de los roles más flexible que la que podemos encontrar normalmente en los libros de texto. "Un rol debe ser flexible, adaptable a las personas que lo representan, los roles son finitos, mientras que las personas son infinitas"[19] , así comienza Jim a describir los diferentes tipos de roles en un proyecto APM, para él, la descripción de los roles no son más que un punto de partida para el producto actual y managers de proyectos, y rápidamente transcienden en roles estáticos. Considera realmente complicado la descripción de los roles debido a la diversidad de formas en que las personas interpretamos cada papel, dando lugar a una riqueza de relaciones inimaginable en nuestras pobres descripciones. Concluye afirmando que las personas no son solo más importantes que los procesos, sino también que los roles.
A pesar de estos conceptos tan idealistas, incluye una breve descripción de los participantes que se dan en un proyecto APM:

- Patrocinador ejecutivo.
  Es la persona (o grupo de personas) que lidera el producto y toma las decisiones clave sobre los objetivos y restricciones del producto.

- Gestor de proyectos (Project manager).
  Es la persona que lidera el equipo encargado de liberar o entregar el producto.

- Gestor del producto (Product manager).
  Es la persona responsable de liderar al equipo responsable de determinar que resultados entregar.

- Ingeniero jefe.
  La persona que guía los aspectos técnicos del producto.

- Gestores.
  Un grupo potencialmente grande de individuos que pueden estar al cargo de organizaciones participantes (entiéndase empresas) y que pueden tener poder de decisión sobre el presupuesto o decisiones técnicas que afecten a los resultados del proyecto.

- Equipo del cliente.
  Individuos, que a tiempo parcial o completo, están encargados de determinar carcaterística que necesitan ser construidas y priorizadas.

- Equipo del proyecto.
  Individuos, que a tiempo parcial o completo, son los encargados de liberar o entregar los resultados.

- Proveedores.
  Compañías externas o personas físicas que proveen servicios o componentes.

- Gobierno.
  Agencias reguladoras que requieren información, reportes, certificaciones y otros documentos administrativos o carácter legal.

3. *Prácticas.*

En el libro *Agile Project Management* [19] especifican las diferentes prácticas que Jim aconseja para cada una de las fases del proyecto explicadas anteriormente, también podemos ver en el anexo 2 un documento que el propio Jim muy amablemente me cedió para evaluar la agilidad de los diferentes proyectos, en el cual enumera diferentes prácticas. Veamos algunas de las prácticas más destacadas:

- Prácticas ágiles iniciando los proyectos y planificándolos. Aquí mostraremos algunas prácticas ágiles que se llevan a cabo a lo largo de las fases de previsión y especulación.

  - La caja del producto como visión. La técnica de vision box o caja de producto como visión es original de Bill Shackelford y consiste en la realización de una imagen visual del producto. En esta actividad el equipo completo se divide en grupos de cuatro a seis personas, su tarea es diseñar la caja del producto. Esto incluye darle un nombre al producto, una imagen gráfica, de tres a cuatro puntos clave para vender el producto, una descripción detallada del producto y requisitos de funcionamiento.

  - Arquitectura del producto. El objetivo de la arquitectura del producto es mostrar las "pesadeces" internas del proyecto, es decir observar internamente que partes del proyecto son más pesadas y por tanto ocuparan más recursos.

  - Hoja de datos del proyecto. Su objetivo es transmitir la esencia, en términos de objetivos, planificación y recursos, de como un proyecto entregará la visión. En otras palabras, la hoja de datos del proyecto es una hoja simple con un resumen de los principales objetivos de negocio, especificación del producto e información de gestión del proyecto.

  - Listado de características del producto. El objetivo de esta practica es expandir la visión del producto, a través de un proceso evolutivo de definición de requisitos del producto, en una lista de requisitos del producto.

  - Tarjetas de especificación de rendimiento. Documentan las principales operaciones y los requisitos de rendimiento del producto que se va a construir.

- Prácticas ágiles obtención de resultados. Corresponde a la fase de exploración.

  - Gestión de la carga de trabajo. El objetivo de esta técnica es tener miembros del equipo que sean capaces de gestionar las actividades diarias requeridas para obtener las características necesarias al final de cada iteración. Exactamente lo que estamos diciendo es que un gestor de proyectos debe monitorizar y no micro-gestionar.

- Coaching y desarrollo del equipo. Esta técnica tiene como objetivo principal incrementar la capacidad del equipo, ayudando a sus miembros a mejorar continuamente sus conocimientos (técnicos y de negocio), su auto-disciplina y habilidades de trabajo en grupo.

- Reuniones diarias del equipo. El objetivo de esta técnica es coordinar las actividades de los miembros del equipo diariamente.

- Decisiones consensuadas y participativas. El objetivo de esta técnica es proveer a los componentes del proyecto con prácticas específicas para elaborar, hacer y analizar el gran número de decisiones que se deben tomar a lo largo del proyecto.

* Práctica ágiles para reflexionar, aprender y adaptarse. Correspondiente a las fases de adaptación y cierre.

    - Revisión y acciones de adaptación del producto, proyecto y equipo. El objetivo de esta técnica es asegurar frecuentes cambios de impresiones (feedback) y que se den altos niveles de aprendizaje en las múltiples dimensiones del proyecto. Con tal de conseguir este objetivo, se realizan reuniones orientadas al cliente, donde se hacen demostraciones del producto final y de esta manera se obtiene la opinión de los clientes. También se realizan revisiones técnicas, evaluaciones del rendimiento del equipo, reportes del estado del proyecto, ...

4. Adopción y experiencias.

Prácticamente esta fue la cuestión por la que tuve que ponerme en contacto con Jim, no existen estudios públicos al respecto de empresas que hayan utilizado esta metodología en sus proyectos. Cuando nos encontramos dentro del mundo empresarial, la mayoría de implantaciones son confidenciales o no les interesa informar de la metodología que están utilizando, por tanto no podemos determinar el grado de adopción, ni experiencias al respecto. Si nos dirigimos a los grupos de correos, si que podemos encontrar referencias a algunas experiencias, pero siempre sin nombres de empresas, de un modo totalmente anónimo.

5. Entorno de uso.

Al ser una metodología que prácticamente se centra en la gestión de proyecto ágiles, su ratio de acción podría considerarse acotado a proyecto con eminentes necesidades ágiles. Pero es interesante la reflexión que hace Jim sobre la gestión de grupos de trabajo muy grandes, aplicando prácticas ágiles, prácticamente cualquier proyecto puede ser gestionado utilizando APM. Es evidente, pero, que es recomendable aplicar esta metodología a proyectos con características ágiles, es decir, en los cuales debemos ir explorando los requisitos, queremos obtener un producto innovador y no podemos aplicar prácticas repetitivas para generar el producto.

6. Estudios actuales.

Agile Project Management goza de una comunidad activa de foros de discusión[20] y de una red de intercambio de conocimiento[21], donde podemos encontrar congresos, conferencias y cursos oficiales que se imparten sobre Agile Project Management.

Crystal Methods.

La familia de metodologías Crystal es un conjunto de diferentes metodologías que podemos seleccionar en función de la adecuación al proyecto en el que nos encontremos. Crystal también incluye un conjunto de principios para adaptar las diferentes metodologías según las circunstancias del proyecto.
Cada una de las metodologías de la familia Crystal tiene asignado un color, cuanto más oscura sea su tonalidad, más compleja es la metodología [22](ver figura 7).

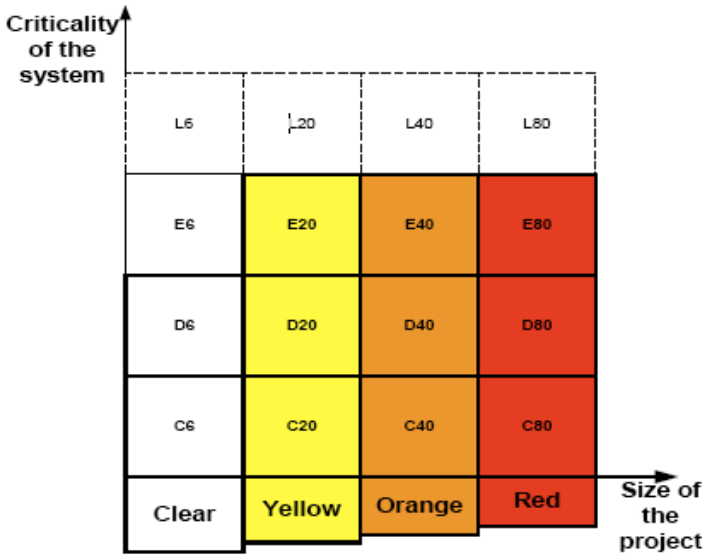


fig 7

Crystal sugiere que escojas el color de la metodología para un proyecto en función de su criticidad y tamaño. Los proyectos más grandes suelen necesitar una mayor coordinación y metodologías más complejas que no los proyectos más pequeños. Cuanto más crítico sea el sistema que queremos desarrollar, más rigurosidad necesitamos disponer en el desarrollo del proyecto. En la figura 16 aparecen unos caracteres (C,D,E y L) e indican las perdidas potenciales por fallos del sistema, y lo hacen de la siguiente manera:

- C, indica pérdida de confort debido a un fallo del sistema.
- D, indica pérdida de dinero discrecional, es decir del que podemos disponer, generalmente nuestro.
- E, indica pérdida de dinero esencial, es decir dinero que probablemente no es nuestro y no podemos disponer de el libremente.
- L, de Life en ingles, vida. Indica la pérdida de vidas por el fallo del sistema.

Las dimensiones de criticidad y tamaño, son representadas por un símbolo de la categoría del producto, que aparece en la figura 16. Por ejemplo, D6 indica un proyecto con un máximo de 6 personas, de máxima criticidad de dinero discrecional.
Las metodologías que integran la familia Crystal tiene en común un seguido de características. Primero, los proyectos siempre usan ciclos de desarrollo incrementales, de una longitud máxima de cuatro meses, siendo preferibles periodos de un mes a tres meses [22]. Segundo, se hace énfasis en la comunicación y la cooperación de la gente. Tercero, las metodologías Crystal permiten el uso de prácticas y herramientas de otras metodologías ágiles como XP o Scrum.
Actualmente existen tres metodologías Crystal que Cockburn asegura que han sido utilizadas en proyectos reales, estas son Crystal clear, Crystal Orange, Crystal Orange Web. También son las únicas metodologías documentadas por Alistair, por lo tanto serán las metodologías a las cuales nos

referiremos a continuación. En realidad son solo dos, ya que Crystal Orange Web es una variante de Orange.

1. *Procesos.*

Crystal Clear esta diseñada para pequeños proyectos, proyectos de categoria D6, pudiendo contar con un equipo de desarrolladores formado por 6 personas como máximo. Algunas modificaciones nos permitirían utilizar Crystal Clear con proyectos de tipo E8 o D10. Dada las limitaciones de comunicación de la estructura, el equipo debería encontrarse ubicado en una oficina común.
Crystal Orange esta diseñada para proyectos de mediana envergadura, pudiendo estar formada desde 10 hasta 40 componentes (categoría D40)y con una duración del proyecto entre uno y dos años. También podemos utilizar Orange con proyectos de categoria E50, si añadimos procesos de verificación de las pruebas. El proyecto se divide en diferentes equipos multifuncionales (utilizando estrategia de diversidad integral que explicamos en prácticas) y se da gran importancia al tiempo conocido como TTM[10].

La familia de metodologías Crystal no especifica ningún ciclo de vida concreto, ni ningún modelo de procesos, en vez de eso lo que hace es determinar una guía de las políticas estándar, productos de trabajo, asuntos locales, herramientas, estándares y roles. Aquí explicaremos un poco por encima cada uno de estos puntos:

- Políticas estándar.
  Son las prácticas necesarias que se aplican a lo largo de todo proceso de desarrollo del software. Tanto Clear como Orange recomiendan las siguientes:

  - Entregas incrementales hechas regularmente.
  - Seguimiento del progreso mediante hitos claves basados en entregas de software, más que en entrega de documentos.
  - Usuario involucrado directamente.
  - Pruebas automáticas regresivas de las funcionalidades.
  - Dos usuarios criticando cada entrega.
  - Talleres del producto y puesto a punto de la metodología, en medio y al final de cada iteración.

- Artefactos.
  Clear y Orange comparten los siguientes ítems: secuencia de entregas, modelado de objetos, manual de usuario, casos de pruebas y código.
  Sin embargo, a modo particular Clear incluye anotaciones de los casos de uso y características del producto y Orange requiere de un documento de especificación de los requisitos. La planificación en Clear se pide que se haga pensando en las diferentes entregas y reuniones con los clientes, mientras que para Orange se requiere una planificación mucho más exhaustiva. Orange exige otros ítems que diferencian los diferentes tipos de proyectos para los cuales están pensados cada una de las metodologías, por ejemplo, Orange pide documentos de diseño de la interfaz de usuario, reportes de estado y especificaciones de los diferentes equipos y Clear no.

- Asuntos locales.
  Son asuntos locales aquellos procedimientos que Crystal especifica que deben hacerse, pero el como, es responsabilidad del proyecto. Por ejemplo, la documentación del proyecto es necesaria, pero como se haga es un asunto local del proyecto.

---

[10] Del inglés Time To Market, tiempo desde que se concibe el producto como una idea, hasta que esta disponible en el mercado.

- Herramientas.
  Las herramientas que necesita la metodología Crystal Clear son un compilador, una herramienta de control de versiones y una herramienta de gestión, ademas de pizarras, que ayudan a sustituir algunos documentos escritos y la realización de reuniones.
  Las herramientas mínimas para la metodología Crystal Orange son aquellas utilizadas para el control de versiones, programar, pruebas, comunicación, seguimiento del proyecto, dibujo y para medir el rendimiento.

- Roles.
  Los veremos en el siguiente punto.

- Actividades y estándares.
  Crystal Orange propone seleccionar las notaciones estándares, convenciones de diseño, estándares de formato y de calidad, a ser usadas en un proyecto.
  Las actividades las veremos en mayor detalle en el punto de prácticas. En la figura 17 podemos ver las principales actividades que se llevan a cabo en una iteración.

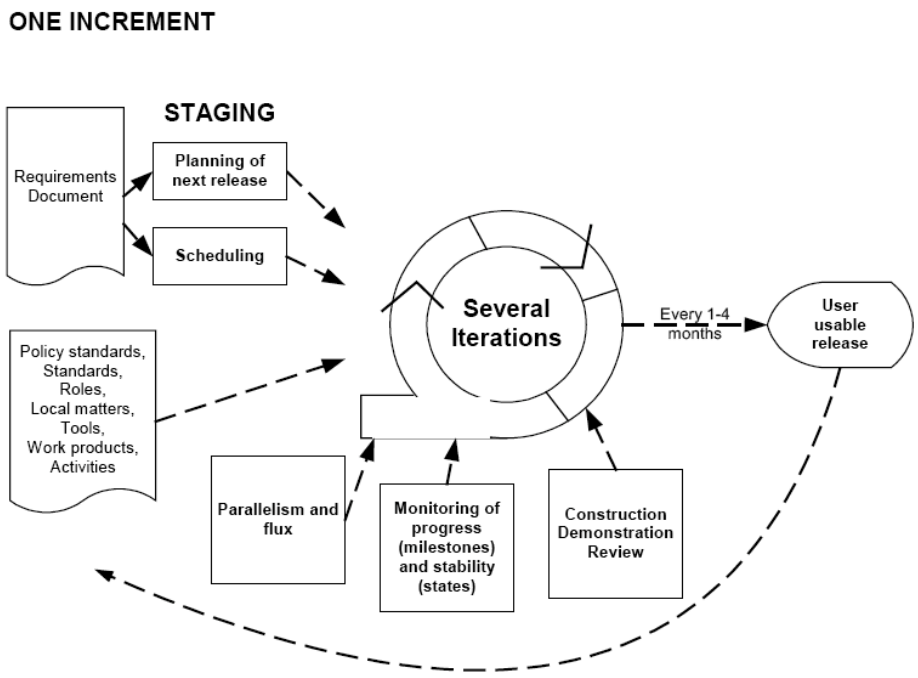


fig 8

2. *Roles y responsabilidades.*

Los roles y responsabilidades que presentaremos a continuación han sido extraídos del *Agile Software Devlopment* [22], tal y como Cockburn los determinó y corresponden a las metodologías Crystal Clear y Orange. Ambas metodologías para cada tarea disponen de múltiples roles, lo que las diferencia es que en Crystal Clear tan solo dispondremos de un equipo, mientras que en Crystal Orange podemos disponer de varios, a la par que introduce algunos roles que no podemos encontrar en Clear.

- Roles de Crystal Clear.

  - Sponsor.
  - Diseñador de programas senior.
  - Diseñador de programas.
  - Usuario. A tiempo parcial como mínimo.

- Roles de Crystal Orange.

  - Sponsor
  - Experto de negocio.
  - Experto de usabilidad.
  - Facilitador técnico.
  - Diseñador/analista de negocio.
  - Gestor de proyecto.
  - Arquitecto.
  - Mentor de diseño.
  - Jefe de diseñadores-programadores.
  - Otros diseñadore-programadores.
  - Diseñador de la interfaz de usuario.
  - Escritor o encargado de documentar.
  - Probador.

Los diferentes roles se pueden agrupar en los siguientes equipos:

  - Planificación del sistema.
  - Monitorización/seguimiento del proyecto.
  - Arquitectura.
  - Tecnología.
  - Infraestructura.
  - Pruebas externas.

Los equipos funcionalmente grandes son divididos en grupos inter-funcionales, utilizando la estrategia de diversidad integra [23]. Cada grupo debe contener un diseñador-analista de negocio, diseñador de interfaces de usuario y de uno a tres diseñadores-programadores. cada grupo también debe tener un diseñador de bases de datos y representantes de otras tecnologías si estas son utilizadas en el proyecto. Cada grupo debe tener también a un probador.

3. Prácticas.

A continuación veremos las prácticas más comunes de las metodologías Crystal, centrándonos en Orange y Clear.

- Planificación por etapas.
  Básicamente consiste en la planificación del siguiente incremento del sistema. La planificación debe finalizar con una versión ejecutable cada tres o cuatro meses como máximo. El equipo selecciona los requisitos que serán implementados en el incremento y planifican lo que creen que serán capaces de hacer.

- Revisiones y resúmenes.
  Cada incremento consta de diferentes iteraciones y cada iteración incluye las siguientes actividades: construcción, demostración y resumen de los objetivos del incremento.

- Monitorización.
  Los progresos del proyecto son monitorizados a partir de las diferentes entregas del equipo durante el proceso de desarrollo. El progreso se mide con los hitos clave y la estabilidad de las fases (my inestable, fluctúa y lo suficiente estable para revisar).

- Paralelismo y flujo.
  Cuando el monitor de estabilidad nos indica un estado lo suficientemente estable para su

revisión, entonces es el momento para pasar a la siguiente tarea. En Crystal Orange esto nos indica que los diferentes equipos pueden trabajar con la máxima eficiencia concurrente (en paralelo). Con tal de poder llevar esto a cabo, los equipos de seguimiento y arquitectura deben revisar sus planes de trabajo, su estabilidad y sincronización.

- Estrategia de diversidad integral.(Holistic diversity strategy)
  Es un método incluido en Crystal Orange y es utilizado para dividir grandes equipos funcionales en pequeños equipos multifuncionales. La idea principal es crear pequeños equipos multidisciplinares, con componentes de múltiples especialidades.

- Técnicas de puesta a punto de la metodología.
  Es una de las técnicas básicas tanto de Crystal Orange como de Crystal Clear. Se basa en la realización de entrevistas y talleres elaborar una metodología específica para cada proyecto. Una de las ideas centrales es modificar o fijar el proceso de desarrollo, es muy importante ver que cada vez que finaliza una iteración, el equipo tiene más experiencia y puede modificar aspectos para que la metodología se adapte mejor al proyecto.

- Punto de vista del usuario.
  En Crystal Clear se sugiere la opinión de dos usuarios por cada versión del producto liberada, mientras que en Crystal Orange se deben hacer hasta tres revisiones por parte del cliente en cada iteración.

4. Adopción y experiencias.

Alistair documenta una experiencia de uso en un proyecto real [23] de métodos y prácticas de la metodología Crystal Orange. El proyecto se conoce como proyecto 'Winifred' y tuvo una duración de dos años, siendo un proyecto de tamaño medio y llegando a contar con un personal de hasta cuarenta personas. Seguía una estrategia de desarrollo incremental y un enfoque orientado a objetos, el objetivo era realizar la migración a un lenguaje orientado a objetos de un sistema heredado para un mainframe.
Alistair identifica problemas desde el primer incremento, problemas con la comunicación, tanto dentro de los equipos como con el "exterior". También se detectó que se había realizado un fase de recogida de requisitos excesivamente larga, que provocó que no se diseñase nada de la arquitectura durante meses. Y otro de los problemas destacados del proyecto era una mala asignación de las tareas. Todo esto propició el uso de la metodología Crystal Orange, las prácticas que se llevaron a cabo fueron la adopción de procesos iterativos para cada incremento, hecho que propicio que los propios equipos se dieran cuenta de sus fallos y tuviesen la capacidad de auto-organizarse. También, las iteraciones incluían vistas funcionales con los clientes, para determinar los requisitos finales, manteniendo a los usuarios involucrados con el proyecto. Las lecciones aprendidas en las primeras iteraciones ayudaron a realizar una mejor distribución de las tareas, establecer mejores líneas de comunicación, ...
Un hecho destacable de este proyecto fue que diferentes equipos de desarrollo, podían estar desarrollando diferentes números de iteraciones, en función de sus tareas.

5. Entorno de uso.

Las metodologías que actualmente están descritas (Orange y Clear) tienen ciertas limitaciones que no les permiten desarrollar proyectos con un alto componente crítico (altos factores de riesgo), de la mismo forma que ambas prescriben su uso a entornos locales. Crystal Clear restringe su entorno de trabajo a un solo equipo, situado en una misma oficina y Orange también requiere que sus equipos se encuentren ubicados en una misma oficina, siendo capaces de tratar tan solo con proyectos de máxima criticidad de dinero discrecional.

6. *Estudios actuales.*

Alistair propuso dentro de su familia de metodologías hasta cuatro metodologías diferentes, de las cuales tan solo ha especificado dos. Alistair se encuentra actualmente desarrollando estas metodologías, y no tenemos noticias de otras entidades, comunidades u organizaciones que estén realizando otras tareas de investigación sobre las metodologías Crystal.

Dynamic Sytems Development Method.

DSDM tiene sus orígenes en 1994 y desde entonces se ha convertido en uno de los frameworks de desarrollo rápido de aplicaciones más utilizados y conocidos. DSDM es un framework sin ánimo de lucro y sin propietarios para el desarrollo rápido de aplicaciones, mantenido por el DSDM Consortium[11].
La idea fundamental de DSDM se basa en que en vez de fijar las funcionalidades de un producto y después el tiempo y el coste, fijar primero el tiempo y el coste y con esto fijado, determinar las funcionalidades que se pueden implementar en el producto.

1. *Procesos.*

DSDM esta formado por las cinco siguiente fases: estudio de viabilidad, estudio de negocio, modelo funcional de las iteraciones, iteraciones de diseño y construcción e implementación (ver figura 9).

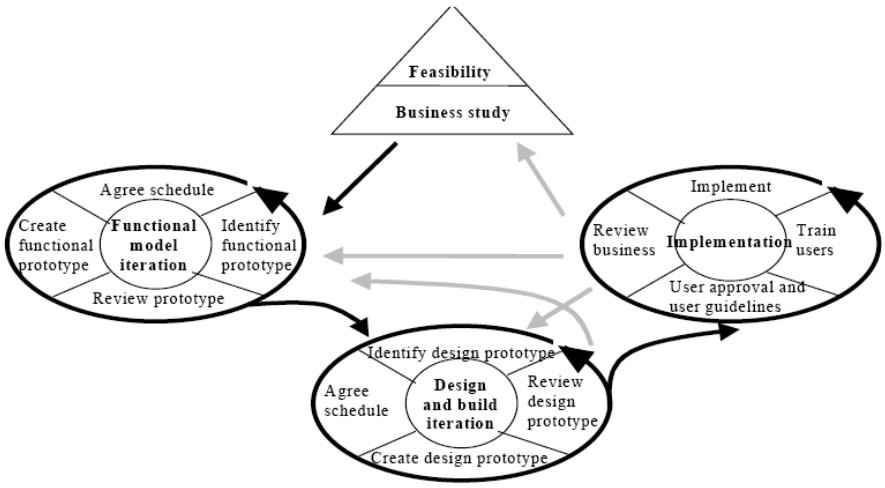


fig 9

Las dos primeras fases son secuenciales y se hacen solo una vez, mientras que las otras tres son iterativas e incrementales a lo largo de todo el proceso de desarrollo. DSDM trata las iteraciones como cajas de tiempo y estas duran un predeterminado periodo de tiempo, una vez finalizadas, las iteraciones también finalizan. El tiempo de duración de una iteración se decide de antes de iniciarla y normalmente va desde unos pocos días, hasta unas pocas semanas.
Veamos con mayor detalles cada una de las fases de DSDM:

- Estudio de viabilidad.
  En esta fase se estudia la idoneidad de utilizar DSDM en el proyecto que nos ocupa y por

11 http://www.dsdm.org/

tanto, se decide si utilizarlo o no. Otros aspectos que se tienen en cuenta en el estudio de viabilidad son las posibilidades técnicas de llevar a cabo el proyecto y los riesgos presentes.
De esta fase se obtienen dos artefactos o productos de trabajo, son un reporte de viabilidad y un esbozo del plan de desarrollo del proyecto. A veces, si la tecnología con la que tratamos es desconocida, se realizar un prototipo para conocerla mejor y ver sus posibilidades. de todo modos, esta fase no debe pasar de unas pocas semanas.

* Estudio de negocio.
  Esta es la fase en que las características principales del negocio y la tecnología, son evaluados. La manera recomendada del levar a cabo esta fase es mediante la realización de talleres, donde participaran los clientes más expertos en la materia, de tal manera que sean capaces de identificar todas las facetas fundamentales del sistema y acordar unas prioridades de desarrollo. Las procesos de negocio y las clases de usuario afectadas se describirán en la definición del área de negocio. Descripciones de alto nivel de los procesos de negocio se incluyen en la definición del área de negocio, ya sea usando diagramas ER[12] u objetos del modelo de negocio.
  Otras artefactos que se generan en esta fase son una definición de la arquitectura del sistema y un esbozo del plan de desarrollo del prototipo.

* Modelo funcional de las iteraciones.
  Es la primera de las fases iterativas e incrementales. En cada iteración los contenidos y el enfoque son planificados, la iteración continua y los resultados son analizados para mejorar las futuras iteraciones. Se realizan las tareas tanto de análisis como de codificación, se construyen prototipos, y la experiencias extraídas de ellos son para mejorar los modelos de análisis. Los prototipos no se descartan por completo, sino que a medida que su calidad va aumentando, se van incluyendo en el sistema final. Se genera un modelo funcional, el cual contiene el código del prototipo y los modelos de análisis. Otra tarea que se realiza en cada una de las iteraciones es la realización de pruebas.
  Otros artefactos que se generan en esta fase son una lista ordenada en función de la prioridad de las funciones que son entregadas al final de la iteración y documentación del prototipo funcional, que incluyen comentarios de los usuarios sobre la iteración actual. Tambié se genera un listado con los requisitos no funcionales y un análisis de los riegos que pueden surgir a lo largo del desarrollo.

* Iteraciones de diseño y construcción.
  Estas son las fases en las que principalmente se construye el sistema. El resultado final es un sistema probado, que como al menos satisface los mínimos requisitos establecidos. Esta fase es iterativa e incremental y el diseño y los prototipos funcionales son revisados por los clientes, ocasionando los cambios que sean necesarios en el sistema con sus aportaciones y comentarios.

* Implementación.
  En esta fase se pasa del entorno de desarrollo al entorno de producción. Se da formación a los usuarios y finalmente se abre el sistema para que lo utilicen.
  En esta fase, además de la liberación del sistema, también se deben entregar un manual de usuario y un informe de revisión del proyecto. En este último entregable, se especifican los futuros desarrollos necesarios, si es que los hay. DSDM define cuatro posibles situaciones una vez en este punto, la primera es que no se necesite realizar mayor desarrollo, la segunda es que se hayan dejado sin realizar un conjunto de requisitos importantes y en este caso se tiene que reiniciar todo el proceso desde el principio. La tercera es que hayan quedado algunas funcionalidades o críticas sin implementar, entonces se vuelve a la fase de modelo

12 Del inglés Entity Relationship, son diagramas de entidades relacionales y son representaciones abstractas y conceptuales de datos estructurados.

funcional de iteraciones. La última situación es que algunos problemas técnicos no han podido ser resueltos debido a falta de tiempo, ahora se corregirán realizando las iteraciones que hagan falta desde la fase de diseño e implementación.

2. *Roles y responsabilidades.*

DSDM especifica hasta quince roles diferentes entre usuarios y desarrolladores, a continuación listamos los más relevantes:

- Desarrolladores y desarrolladores senior. Son los únicos roles que establece a nivel de desarrollo y cubren todos los posibles papeles de las tareas de desarrollo como son el análisis, diseño, implementación e incluso pruebas.

- Coordinador técnico. Es el responsable de la calidad técnica del proyecto y define la arquitectura del sistema.

- El usuario "embajador". Sus principales tareas son las de aportar el conocimiento de la comunidad de usuarios al proyecto y de informar a los usuarios de los progresos del proyecto.

- El "asesor" de los usuarios. Puede ser cualquier usuario que represente un importante punto de vista para el proyecto. Por ejemplo un miembro del departamento de IT o un auditor financiero, etc.

- Visionario.
  Es un usuario que participa del proyecto y que tiene una percepción mas real de los objetivos de negocio del sistema y del proyecto. Normalmente suele ser el valedor del proyecto y el que tuvo la idea inicial para realizarlo. Entre sus tareas se encuentra supervisar que se identifican todos los requisitos de negocio y que se van implementando en el orden de importancia correcto.

- Sponsor ejecutivo.
  Es la persona de la compañía que tiene la autoridad financiera y la responsabilidad. Por ende, el sponsor ejecutivo es el que suele tener la última palabra en la toma de decisiones.

3. *Prácticas.*

Existen nueve prácticas que definen la ideología y las bases de toda actividad en DSDM, en la tabla 7 mostramos estas nueve prácticas o principios de DSDM. Ver siguiente página.

Prácticas	Descripción
La involucración activa del usuario es imperativa.	Unos pocos usuarios experimentados deben seguir todo el proceso de desarrollo para asegurar una valoración correcta y a tiempo.
Los equipos DSDM deben tener libertad poder para tomar decisiones.	En rápidos ciclos de desarrollo no se pueden tolerar largos periodos de decisión. Los usuarios involucrados tienen el conocimiento para decidir hacia donde se dirige el proyecto.
El objetivo es la entrega frecuente de productos.	Las decisiones erróneas pueden ser solventadas si los ciclos son cortos y los usuarios proporcionan unas valoraciones correctas.
Adecuación a los objetivos de negocio es el criterio esencial para aceptar las entregas.	Si los objetivos de negocio aún no han sido satisfechos no podemos dedicarnos a mejorar técnicamente el sistema.
El desarrollo iterativo e incremental es necesario para convergir en una solución de negocio correcta.	Rara vez los requisitos no varían a lo largo del proyecto, si dejamos al sistema evolucionar a través del desarrollo iterativo, los errores se pueden corregir y encontrar antes.
Todos los cambios durante el desarrollo son reversibles.	A lo largo de todo el proyecto es sencillo tomar un camino incorrecto, pero con cortas iteraciones y asegurando que todos nuestro pasos se puedan deshacer, los caminos incorrectos pueden ser rectificados.
Los requisitos son la línea de base en alto nivel.	Congelar los requisitos se puede hacer solo a alto nivel, para que los detalles de los requisitos puedan variar libremente. De este modo fijamos los requisitos esenciales en etapas tempranas del proyecto.
Las pruebas se integran a lo largo del ciclo de vida.	Todos los componentes del sistema deberían ser probados por los desarrolladores y usuarios al mismo tiempo que son desarrollados.
Un enfoque colaborativo y cooperativo compartido por todos los stakeholders es esencial.	Las decisiones de lo que es entregado y de lo que falta por realizar es siempre un compromiso y requiere aceptación por todas las partes que participan.

tabla 7

4. *Adopción y experiencias.*

DSDM ha sido ampliamente utilizado en Reino Unido desde mediados de los años 90. Stapleton [24] ha documentado hasta ocho casos de aplicación práctica de DSDM, demostrando que DSDM es una alternativa viable para el desarrollo rápido de aplicaciones.
Con tal de facilitar la adopción de DSDM, el DSDM Consortium publicó un método que indica la idoneidad de aplicar DSDM a tu proyecto. Si quiere obtener más información sobre este método diríjase *Dynamic System Development Mehtods de Stapleton*[24].

5. *Entorno de uso.*

El tamaño de los equipos en DSDM varia desde un mínimo de dos integrantes hasta seis, pudiendo coexistir múltiples equipos en un mismo proyecto. La razón de que como mínimo hayan dos componentes por equipo radica en que como mínimo deben haber un desarrollador y un cliente. El número máximo ha sido extraído de diferentes experiencias con la metodologías. DSDM se puede aplicar tanto a proyectos grandes como pequeños, siempre que los sistemas grandes sean divisibles en componentes que puedan ser desarrollados por equipos pequeños. Stapleton [24] sugiere la utilización de DSDM en aplicaciones empresariales antes que en aplicaciones científicas.

6. *Estudios actuales.*

DSDM fue originalmente creada por un consorcio y actualmente sigue gestionada y mantenida por el mismo, el cual esta compuesto por diferentes compañías. Si quieres poder acceder a la información que este consorcio te proporciona, sus manuales, white papers, avances, etc, es necesario que pagues un cuota de asociado.

Scrum.

La primera vez que se asocio el término Scrum a a los procesos de desarrollo fue en en 1986, cuando Nonaka y Takeuchi presentaron su artículo *The New Product Development Game*[25]. Nonaka y Takeuchi presentaban en este artículo un proceso adaptativo, rápido y auto-organizado de desarrollo de productos. El término Scrum deriva del mismo término en rugby, que hace referencia a como se devuelve un balón que ha salido fuera del campo, al terreno de juego de una manera colectiva, la traducción al castellano sería melé.
Scrum surgió como práctica en el desarrollo de producto tecnológicos y no sería hasta 1993 que Jeff Sutherland aplicará el modelo al desarrollo de software en la Easel Corporation [1]. En 1996 Sutherland presentó junto con Ken Schwaber las practicas que empleaba como procesos formal para la gestión del desarrollo de software en OOPSLA 96. Estas practicas de gestión pasarían a incluirse junto con otras muchas en la lista de modelos ágiles de Agile Alliance en el año 2001.

1. *Procesos.*

Podemos identificar tres fases en Scrum, planificación del sprint, seguimiento del Sprint y revisión del sprint (ver figura 10[13]). Schwaber llamaban a estas tres fases como fase antes del juego, fase del juego o desarrollo y fase de después del juego, pero tienen algunos matices diferentes en los que no entraremos.

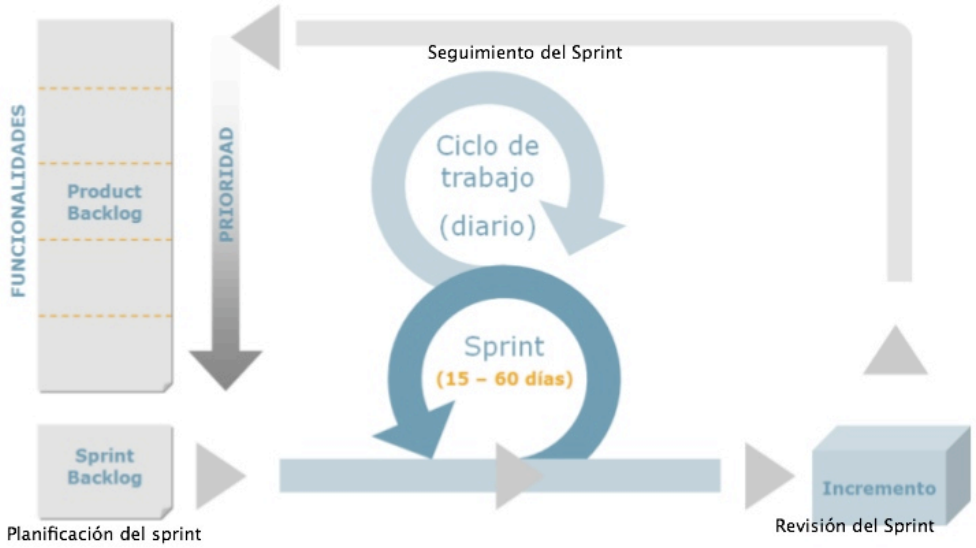


fig 10

[13] Figura obtenida de Flexibilidad con Scrum [25], y añadidas pequeñas modificaciones.

Veamos las principales tareas que se llevan a cabo en cada fase:

- Planificación del Sprint.
  En esta fase se define el Product Backlog (ver prácticas de Scrum), si todavía no ha sido definido, el cual consiste en una lista priorizada de requisitos del sistema y es un documento vivo, que puede ser continuamente actualizado. En cada iteración el product Backlog es revisado por el equipo. También se lleva a cabo la planificación del primer Sprint[14]. La planificación de cualquier sprint es la jornada de trabajo previa al inicio de cualquier sprint y en la cual se determinan cuales son los objetivos y el trabajo que se deben cubrir en esa iteración. En esta reunión se obtiene una lista de tareas que denominamos sprint backlog, y el lema u objetivo principal del sprint.

- Seguimiento del Sprint.
  A lo largo de esta fase se llevan a cabo breves reuniones diarias, para ver el avance de las tareas y el trabajo que esta previsto para la jornada. En estas reuniones solo están presentes el Scrum Master y el equipo y las preguntas que se realizan suelen ser tres [26]:

  - Que trabajo se ha realizado desde la reunión anterior.
  - Trabajo que se va ha hacer hasta la próxima reunión.
  - Impedimentos que deben solventarse para proseguir con el trabajo.

- Revisión del Sprint.
  Una vez finalizado el Sprint se realiza un análisis y revisión del incremento generado. En esta reunión se presentan los resultados finales y se recomienda siempre tener preparada una demo. Existen múltiples razones para recomendar tener una demo al final de cada sprint, entre ellas destacamos la mejora del feedback con los interesados, reconocimiento del trabajo, un esfuerzo por finalizar las cosas o un correctivo en caso de tener una demo mala.

2. *Roles y responsabilidades.*

Segun Schwaber y Beedle existen seis tipos de roles diferentes en la metodología Scrum y son:

- Propietario del producto.
  Es la única persona del proyecto conocedora del entorno de negocio del cliente y de la vision del producto y es el responsable de obtener el resultado de mayor valor posible para el cliente. También es el responsable de la financiación necesaria para el proyecto, de tomar las decisiones que afecten a como va a ser el resultado final, fechas de lanzamiento y el retorno de inversión. Por regla general y si no se trata de proyectos internos, el propietario del producto suele ser el responsable del proceso de adquisición del cliente. Sino, el product manager también puede asumir este rol.

- Scrum Master.
  Es el encargado de garantizar el funcionamiento de los procesos y de la metodología. Es importante darse cuenta que Scrum master es más que un rol, es la responsabilidad de funcionamiento de modelo, por tanto muchas veces es aconsejable utilizar a personas y puestos más adecuados según la organización.
  Un Scrum Master debe interactuar tanto con el equipo como con el cliente y con los gestores.

- Equipo de desarrollo.
  Es el equipo del proyecto y tiene la autoridad para decidir en las acciones necesarias y para auto-organizarse con la finalidad de alcanzar os objetivos del sprint.

---

14 En Scrum a las iteraciones se les llama Sprints.

El equipo esta involucrado en la estimación del esfuerzo de las tares del product backlog, en la creación del sprint backlog, etc.

- El cliente.
  El cliente participa en la creación del product backlog.

- El gestor.
  Esta al cargo de la toma de decisiones finales, participa en la elección de objetivos y requisitos. Una de las actividades en las que esta involucrado el gestor es en la selección del propietario del producto o en la reducción del product backlog junto con el Scrum Master.

3. *Prácticas.*

Scrum no requiere y/o provee de ninguna práctica concreta para el desarrollo del software, sin embargo si que dispone de prácticas y herramientas para la gestión de las diferentes fases de Scrum. A continuación mostramos las principales prácticas y herramientas:

- Product Backlog.
  Define los requisitos del sistema o el trabajo ha hacer a lo largo del proyecto. Esta compuesto por una lista de requisitos de negocio y técnicos, actualizados y priorizados.
  El responsable de mantener el product backlog es el propietario del producto.

- Sprint Backlog.
  Es una lista de trabajos que el equipo se compromete a realizar para generar el incremento previsto. Las tareas están asignadas a personas y tienen estimados el tiempo y los recursos necesarios.

- Estimación de esfuerzo.
  Es un proceso iterativo en el cual las estimaciones de los ítems del product backlog son reajustadas acorde a la información obtenida en la última iteración. Este reajuste lo llevan a cabo el equipo de desarrollo y el propietario del producto.

- Gráfico Burn-down.
  Es una herramienta para gestionar y seguir el trabajo de cada sprint y representa gráficamente el avance del sprint. Si observamos la figura 11[27] (ver página 106) podremos observar en el eje horizontal una escala temporal que representa los días de duración del sprint y en el eje vertical otra escala que indica los puntos de historia o estimación de esfuerzo que hemos comentado antes. De esta manera en las reuniones diarias el Scrum Master puede evaluar cuanto se ha avanzado y actualizar el gráfico.

- Gráfico Burn-up.
  Herramienta de gestión y seguimiento que sirve al propietario del producto para controlar las versiones de producto previstas, las funcionalidades de cada una, la velocidad estimada, fechas probables de cada versión, margen de error previsto en las estimaciones y avance real.
  Herramienta de gestión y seguimiento para el propietario del producto.

- Planning Poker o estimación de Poker. Juego que ayuda al equipo a establecer una estimación de las tareas en la reunión de inicio de sprint.

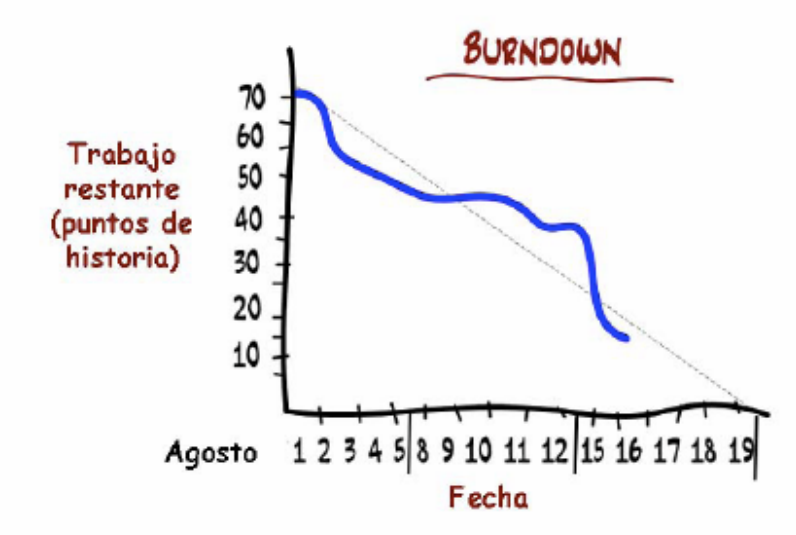


fig 11

4. *Adopción y experiencias.*

Schwaber y Beedle [26] identifican dos tipos de situaciones en las que se puede adoptar Scrum, para proyecto nuevos o para proyectos existentes. En los proyectos existentes, la tecnología a ser usada y el entorno ya existe, pero el equipo de desarrollo se esta encontrando con problemas de requisitos variables y tecnologías complejas. En esta situación se recomienda iniciar las reuniones diarias con un Scrum Master y en el primer sprint tener un funcionalidad implementada con la tecnología que esta dando problemas, con e objetivo de subir la moral a los desarrolladores y el a todo el equipo en general.
En el caso de ser un proyecto nuevo, el proceso que se seguira será exactamente el que hemos descrito en anteriormente, empezando con un product backlog, priorización de las tareas, planificación del primer sprint, ejecución, ...
Scrum es una metodología que goza de gran salud y de la cual podemos encontrar bastantes empresas que gustan de gritar a los cautro vientos que utilizan Scrum, entre ellas podemos encontrar empresas tan importante como Yahoo o Google. En el año 2000 Rising and Janoff publicaron tres proyectos exitosos utilizando Scrum.[29]

5. *Entorno de uso.*

Scrum es un método ideal para pequeños equipos, de diez o menos componentes. Schwaber and Beedle recomiendan equipos de cinco componentes [26], dividiendo los equipos en equipos más pequeños si fuera necesario.

6. *Estudios actuales.*

Scrum es una metodología que cubre a la perfección las áreas de gestión de proyecto ágiles, de forma similar a Agile Project Management. Parece ser que la mayoría de los esfuerzos se están centrando en combinar metodologías ágiles que sean complementarias, como por ejemplo Scrum y XP[28]

Test Driven Development.

Test Driven Development, TDD o desarrollo orientado a las pruebas, es la técnica por antonomasia de Extreme Programming, tal y como veremos en las siguiente sección. Pero tal es su importancia, que mucha gente ya la considera una metodología independiente, un punto y aparte después de XP. Realmente podemos considerarla una metodología ya que presenta un conjunto de prácticas y métodos de desarrollo, además de que condiciona la mentalidad de los desarrolladores guiándolos a través del desarrollo y aumentando la calidad del producto final.

1. Procesos.

Según David Astels [32] TDD es un estilo de desarrollo donde mantienes un juego de pruebas del programador exhaustivo, ninguna parte del código pasa a producción a no ser que pase sus juegos asociados, escribes primero las pruebas y las pruebas determinan el código que necesitas escribir. Veamos en mayor detalle estos principio de TDD:

- Mantener un juego exhaustivo de pruebas del programador.
  Las pruebas del programador prueban que las clases se comporten de la forma esperada, son escritas por los desarrolladores que escriben el código que será probado. Se llaman pruebas del programador porque aunque se parecen mucho a las pruebas unitarias, se escriben por diferentes motivos. Las pruebas unitarias se escriben para demostrar que el código que tu has escrito funciona, las pruebas del programador son escritas para definir que significa que el código funciones. También se llaman así para diferenciarlas de las pruebas escritas por los usuarios.
  Si utilizas Test Driven Development, dispones de un gran juego de pruebas, esto es así porque no puede haber código sino existen las pruebas que los prueben. Primero escribes las pruebas y luego el código que será probado, no hay por tanto código sin pruebas, concluyendo que las pruebas son por tanto exhaustivas.

- Todo código que pasa a producción tiene sus pruebas asociadas.
  Esta característica nos recuerda una propiedad fundamental cuando utilizamos TDD y es que una funcionalidad no existe hasta que tenemos un juego de pruebas que vaya con el. Esta propiedad no brinda la oportunidad de utilizar un par de técnicas muy comunes como son el refactoring y la integración continua. Ambas técnicas solo pueden ser ejecutadas si realmente estamos seguro de que nuestro código sigue cumpliendo una características definidas en las pruebas.

- Escribir las pruebas primero.
  Tal y como estamos comentado en cada punto anterior, cuando tenemos una nueva funcionalidad, antes de implementarla debemos escribir sus pruebas. Esta es una de las características mas “extremes” de TDD. La manera de proceder es escribir un poco de las pruebas y entonces, escribir un poco de código que las ejecute y las pase, entonces otra vez ampliamos las pruebas, volvemos a escribir código, y así sucesivamente.

- Las pruebas determinan el código que tienes que escribir.
  Estamos limitando la escritura de código a tan solo la prueba que ya tenemos implementada. Solo escribes lo necesario para pasar la prueba, esto quiere decir que haces la cosa más sencilla para que funcione.

Test Driven development como tal, se engloba dentro del ciclo de vida de la metodología Extreme Programming, en las fases de iteración, producción y mantenimiento (ver siguiente apartado sobre Extreme Programming), pero en si ya es una metodología, ya que puede ser aplciada independientemente a XP, en proyecto con Scrum, FDD o metodologías tradicionales. Debido a su

radical planteamiento a la hora de escribir código, cambia drásticamente la mentalidad de cualquier equipo de desarrollo, generalmente agilizando los resultados y aumentando la calidad del sistema.

2. *Roles y responsabilidades.*

Normalmente nos encontraremos con los roles típicos de un equipo ágil y con muchas suerte de XP. Estos son los dos roles que como mínimo debe disponer todo proyecto con TDD:

- Cliente.
  Desarrolla las historias con los desarrolladores, las ordena según la prioridad y escriba las pruebas funcionales.
- Desarrollador.
  Desarrolla las historias con el usuario, las estima, y entonces toma responsabilidad de su desarrollo utilizando TDD, lo que quiere decir que ejecuta las pruebas, implementa y refactoriza.

3. *Prácticas.*

Algunas de las prácticas y métodos que voy a mencionar a continuación forman parte de la metodología XP, pero que están fuertemente vinculados con TDD, ya que sin este o no podrían realizarse o bien sería una autentica locura. En concreto estoy hablando de refactoring e integración continua.

- Refactoring.
  Refactoring es el proceso de aplicar cambios a un código existente y funcional sin modificar su comportamiento externo, con el propósito de mejorar la estructura interna y aumentar su legibilidad y entendimiento.
  Esta muy relacionado con TDD ya que muchas veces duplicamos código e introducimos mucha "basura" cuando estamos intentando programar algo para que pase unas pruebas específicas. Otro factor es que si no estamos haciendo TDD, es poco recomendable refactorizar el código, ya que es probable que modifiquemos algo que finalmente provoque un error no intencionado.
  Se suele hacer refactoring cuando existe código duplicado, cuando percibimos que el código no esta lo suficientemente claro o cuando parece que el código tiene algún problema.

- Integración continua.
  Según Martin Fowler[35], la integración continua es una práctica de desarrollo del software donde los miembros de un equipo integran su trabajo frecuentemente, normalmente cada persona integra como mínimo una vez al día, teniendo múltiples integraciones al día. Llamamos integración a la acción de incluir una porción de código a la totalidad del código fuente de la aplicación y normalmente situado en un repositorio de código.
  Cada integración es verificada mediante una construcción automática del mismo (incluyendo pruebas), para detectar errores de integración lo antes posible. Esta práctica reduce los problemas de integración y permite al equipo desarrollar el software cohesivamente más rápido.
  De mismo modo que con la técnica de refactoring, esta íntimamente ligada a TDD, ya que cada vez que se realiza una integración un juego de pruebas exhaustivo se ejecuta y valida el código introducido. Sin estas pruebas, la integración continua no tendría sentido, ya que no garantizaría ni la cohesión, ni la validez del código integrado.

A continuación mostraremos algunas herramientas fundamentales en el uso de TDD, sin ellas sería imposible realizar las tareas que nos proponen.

- Pruebas unitarias.
  En los últimos años TDD ha popularizado el uso de la familia de herramientas xUnit, que

son herramientas que te permiten la ejecución de pruebas unitarias de forma automática.

- Repositorios de código.
  Es una herramienta obligatoria para el uso de la metodología TDD. Las más conocidas son CVS o Subversion y facilitan el control de versiones, acceso e integración de código.

- Software de integración continua.
  Existen diferentes aplicaciones web como Hudson, Apache Gump, Cruise Control, etc, el cometido de las cuales es construir la aplicación a partir del código fuente, normalmente situado en un repositorio de código (subversion,cvs,...) y ejecutar las pruebas especificadas. Si las pruebas son satisfactorias, se despliega en el entorno de producción, en caso contrario no se incluyen las modificaciones. Este es un proceso que se puede ejecutar varias veces a lo largo del día y garantiza que se realiza la técnica de integración continua.

- Herramientas de construcción automáticas.
  Ayudan y mucho la utilización de herramientas como maven o Ant para la compilación y ejecución de las pruebas automáticamente.

4. Adopción y experiencias.

Cuando queremos adoptar una nueva de metodología de trabajo, al principio suele ser duro y TDD no es una excepción. En una entrevista al Dr. Hakan Erdogmus, editor jefe de IEEE Software[36], sobre TDD y partiendo del estudio hecho por Ron Jeffries y Grigori Melnik, publicado como *TDD--The Art of Fearless Programming*, habla sobre muchos malentendidos sobre TDD y los problemas su adopción. Hakan comenta que TDD a veces es entendido como un procedimiento de aseguración de lo calidad, provocadno que algunos managers no lo utilicen porque creen que sus equipos ya tienen otros procedimientos que garantizan la calidad. Hakan incide en que originalmente TDD fue pensado como una técnica para mejorar la productividad y que el aumento de la calidad es un efecto secundario. En cuanto a otros problemas o resistencias a la hora de adoptar TDD, especifica que muchos buenos desarrolladores se resisten a utilizar TDD, simplemente porque no están dispuestos a cambiar sus hábitos de programación. Otros impedimentos en la adopción de TDD que comenta Hakan es la dificultad de practicar TDD a modo personal, en un equipo que no lo usa. También indica que la percepción de los managers o gestores de proyectos, de la utilización de TDD, como un tiempo adicional que podría ser utilizado en otras actividades necesarias, no ayuda en exceso en su adopción.

Podemos encontrar muchas experiencias en las cuales se ha utilizado TDD como parte de Extreme Programing, es más difícil encontrar experiencia en las cuales se documente la utilización de TDD como metodología aislada, normalmente aparece siempre complementando a otra metodología[33][34].

5. Entorno de uso.

Una condición sinequanum de la utilización de TDD, es que todo el equipo lo haga, no se puede utilizar TDD como una práctica aislada, ya que no tendría ningún tipo de valor.
No existe ninguna

6. Estudios actuales.

Test Driven development es una de las metodologías con mayor acogida en el campo profesional y que continua expandiéndose debido a sus buenos resultados. La tendencias actual es integrar TDD independientemente en cualquier metodología ya sea ágil[33] o tradicional[34] y aprovechar los beneficios de practicar una metodología que siempre permite deshacer los errores, asegurar una calidad del producto y protegerse de errores tanto malintencionados como humanos.

Extreme Programming.

Extreme Programming es sin duda el abanderado de las metodologías ágiles. Nació como un intento, bastante exitoso, de establecer un conjunto de prácticas que facilitasen la finalización de los proyectos. Después de unas cuantas exitosas pruebas, estas prácticas se plasmaron de forma teórica[30], dando lugar a una metodología, que mantenía sus principales principios y prácticas.
El término extreme es debido a que las prácticas que se utilizaron, fueron llevadas hasta el extremo.

1. Procesos.

El ciclo de vida de XP esta compuesto de seis fases, ver figura 12.

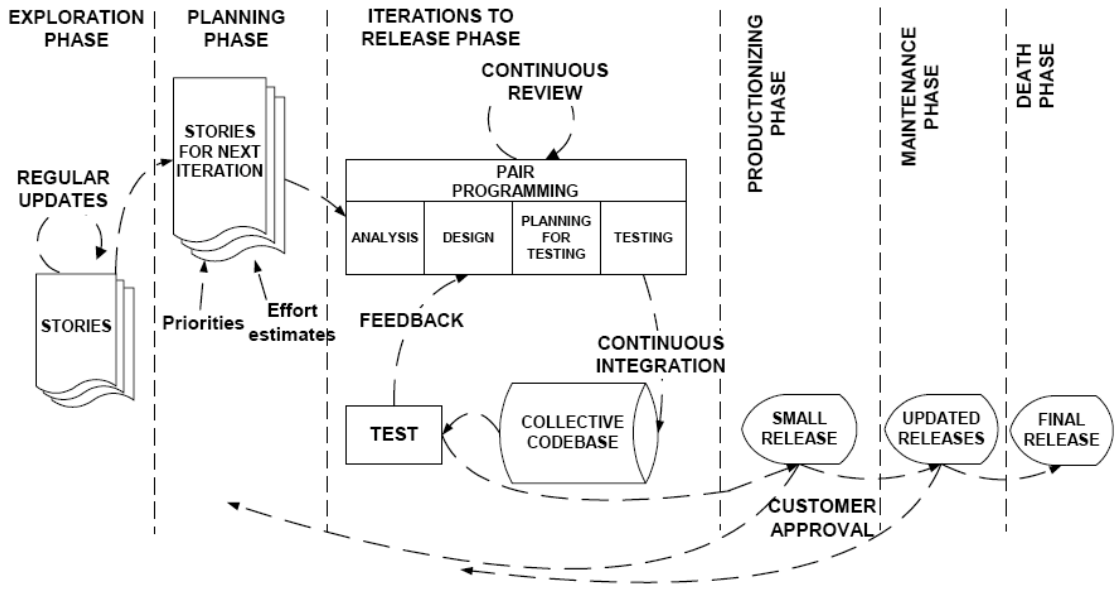


fig 12

- Fase de exploración.
  En esta fase los usuarios escriben las tarjetas de historia que ellos quieren que sean incluidas en la primera versión. Cada una de las tarjetas de historia describen una funcionalidad que será añadida al programa. El equipo de proyecto durante este tiempo se dedica a familiarizarse con las tecnologías y herramientas que utilizará a lo largo del proyecto, probando las herramientas y construyendo un prototipo simple para probar las posibilidades de la arquitectura. El periodo de tiempo de esta fase puede variar desde unas pocas semanas hasta unos pocos meses, dependiendo de la familiaridad del equipo con las tecnologías.

- Fase de planificación.
  En esta fase se establece la prioridad de las diferentes historias y se acuerda el contenido de la primera entrega del proyecto. La estimación temporal se basa en un cálculo estimado por parte de los desarrolladores de cada una de las historias, la primera entrega no suele tardar más de dos meses en realizarse. La duración de esta fase no suele exceder el plazo de unos pocos días.

- Fase de iteraciones.
  Esta fase incluye la realización de diferentes fases antes de liberar la primera versión del producto. La planificación realizada en la etapa anterior se divide en diferentes iteraciones, de una duración variable entre una semana y cuatro. Los usuarios son los que deciden que historias se van a realizar en cada iteración, sabiendo que en la primera se suele realizar un

sistema con la arquitectura de todo el sistema, seleccionando aquellas historias que ayuden a construirla. Las pruebas funcionales creadas por el cliente son ejecutadas al final de cada iteración, de tal manera que al final de esta fase obtenemos una versión lista para producción.

- Fase de producción.
  En esta fase se llevan a cabo se llevan a cabo un conjunto de pruebas extras, de rendimiento y funcionamiento que son necesarias antes de poder entregar el producto al cliente. Si se encuentran cambios importantes que se deban realizar al producto, se debe decidir si incluirlos en esta versión o dejarlos para posteriores. Las iteraciones de esta fase no deben durar más de tres semanas.

- Fase de mantenimiento.
  Una vez se ha liberada la primera versión a los usuarios, el proyecto se debe mantener en el entorno de producción siempre y cuando aún hayan iteraciones en fase de producción. Esto supone un esfuerzo considerable en la fase de mantenimiento e incluso se sugiere la contratación de nuevo personal para dar soporte a los clientes e incluso cambiar la estructura del equipo.

- Fase de cierre del proyecto.
  Es la fase en que los clientes ya no tienen más historias que deban ser implementadas. Es necesario para que podamos considerar que estamos en esta fase, que se satisfagan todas las necesidades de los clientes y otros aspectos como fiabilidad, rendimiento, etc. La documentación del proyecto se realiza en esta fase, ya que ni la arquitectura, ni el diseño, ni el código sufrirán cambio alguno. También podemos encontrarnos en esta fase si las historias que se desean implementar tienen un coste demasiado elevado para su desarrollo.

2. Roles y responsabilidades.

A continuación presentamos los diferentes roles que podemos observar en un proyecto XP, según las tareas que desarrollan y los propósitos que persiguen:

- Cliente.
  El cliente es el encargado de escribir las historias y las pruebas funcionales, y es el que decide cuando un requisito es satisfecho. También es el encargado de establecer la prioridad de las funcionalidades a implementar.

- Programador.
  Escribe tanto las pruebas como el código de la aplicación y se debe comunicar fluidamente y frecuentemente con sus compañeros.

- Probador.
  Ayuda a escribir las pruebas funcionales al cliente, ejecuta regularmente las pruebas funcionales, transmite los resultados de las pruebas y mantiene las herramientas.

- Rastreador.
  Es el encargado dar el "feedback" en XP. Se encarga de seguir las estimaciones hechas por los clientes y de ir avisando de las desviaciones posibles y como cuanto se ajustan a la realidad, con tal de mejorar las futuras estimaciones. También se encarga de realizar el seguimiento de las iteraciones y valorar si los objetivos se pueden alcanzar con los recursos disponibles (tiempo, personal) o si es necesario hacer algún cambio.

- Coach o tutor.
  Es la persona responsable de todo el proceso. Es importante que tenga conocimientos y experiencia en otros proyectos de XP y de este modo guiar y ayudar al equipo a adaptarse a

XP.

- Consultor.
  Es un miembro externo que tiene los conocimientos técnicos necesarios. El consultor guía al equipo para solucionar los problemas específicos.

- Gestor o Manager. Más conocido como el jefe.
  Es el que toma las decisiones críticas y esta en permanente contacto con el equipo para poder discernir las diferentes situaciones críticas.

3. *Prácticas.*

Extreme Programming dispone de un gran abanico de técnicas y prácticas, muchas de ellas que han sido escogidas de diferentes metodologías existentes que tenían suficientemente probada su eficiencia. Veamos a continuación las principales técnicas y prácticas de XP:

- El juego de la planificación.
  El equipo de desarrolladores estiman el esfuerzo necesario para implementar las historias y los clientes determinan los objetivos y tiempos de entrega.

- Historias de usuario.
  Son los requisitos del sistema formulados en una o dos sentencias, en el lenguaje común del cliente.

- Cortas y pequeñas iteraciones.
  Un sistema simple se libera cada dos o tres meses y las diferentes versiones del mismo se suceden en periodos no superiores al mes.

- Metáforas.
  El sistema se define utilizando un conjunto de metáforas acordadas entre el cliente y los programadores. Esta historia compartida guiará todo el proceso describiendo como funciona el sistema.

- Diseño simple.
  Se da gran importancia a la obtención de diseños simples que se puedan implementar rápidamente, evitando diseños complejos y código extra.

- Pruebas.
  El desarrollo del software es orientado a pruebas (Test driven development) Las pruebas unitarias se escriben antes que el código y están funcionando continuamente. Las pruebas funcionales las escriben los clientes.

- Refactorizar.
  Reestructurar el sistema eliminando duplicaciones, mejorando la comunicación, simplificando y añadiendo flexibilidad.

- Programación por pares.
  Es la técnica que promulga que dos personas escriban código en el mismo ordenador.

- Propiedad colectiva.
  Cualquiera puede compartir cualquier parte de código con cualquier otro componente del equipo.

- Integración continua.
  Una nueva porción de código es integrada en el código fuente, tan pronto como este lista. El

sistema es integrado y construido muchas veces a lo largo del día, todas las pruebas son ejecutadas y deben ser pasadas para aceptar la nueva porción de código.

- 40 horas a la semana.
  Se establece un máximo de 40 horas de trabajo semanales.

- Disponibilidad del cliente.
  Los clientes deben estar disponibles y presentes cuando sean requeridos por el equipo de desarrollo.

- Estándares de codificación.
  Reglas de codificación son establecidas y seguidas por los programadores. Se enfatiza la comunicación a través del código.

- Espacios de trabajo abiertos.
  Un gran espacio, con cubículos es lo recomendable.

4. *Adopción y experiencias.*

Una de las referencias más comunes que se hacen tanto a Kent Beck como a XP es una frase que Beck escribió en *Extreme Programming Explained*[30] y que hace referencia a como es recomendable adoptar XP, "*Si quiere probar XP, por el amor de Dios no lo intentes de una sola vez. Escoge tu peor problema en tu proceso actual e intenta solucionarlo de un modo XP*". Beck expresa claramente que XP es difícil de implementar de una sola vez y es recomendable ir escogiendo con cuidado que prácticas aplica a tu proyecto, poco a poco, hasta que finalmente tengas el suficiente criterio para utilizarlo de una forma más libre.
Cada práctica debe ser adaptada a los diferentes proyectos y no adaptar tu proyecto a las prácticas que hemos explicado anteriormente, pero, ¿cuanto puedes modificar una práctica y que siga siendo parte de la metodología XP? En realidad no existen o no he encontrado referencias a proyectos que hayan implementado todas y cada una de las prácticas de XP, pero si podemos encontrar diferentes experiencias de adopciones parciales.[31]

5. *Entorno de uso.*

Beck[30] cree que a pesar de ser una metodología ampliamente utilizada, tiene sus límites, los cuales aun no han sido claramente establecidos. Beck continua especificando que XP es una metodología en la que es recomendable trabajar con equipos de tamaño medio o pequeños, es decir entre tres y veinte componentes como máximo. Considera intolerable esparcir los componentes de un mismo equipo en ubicaciones distintas que no sean en la misma planta y oficina. También considera un factor determinante la cultura de la empresa, declarando que si observamos resistencia en cualquiera de las prácticas o técnicas de XP es muy probable que el proyecto fracase. Otro factor clave para utilizar con éxito XP es la tecnología, si esta no acepta cambios frecuentes o requiere de continuo feedback, es muy probable que no consigamos sacar el proyecto adelante.

6. *Estudios actuales.*

XP ha demostrado ser la metodología que más estudios dispone y mayor número de papers se han escrito sobre ella (ver tabla 5 de la sección resultados del estudio). A diferencia de las otras metodologías, los estudios que podemos encontrar son en su mayoría de casos prácticos, más que estudios teórico o académicos.

Conclusiones

A lo largo de este capítulo hemos podido observar el proceso de transformación que ha sufrido el desarrollo del software a lo largo del siglo pasado e inicios del actual. Nos hacemos llamar ingenieros del Software, pero todavía no sabemos como controlar nuestros proyectos, tampoco es que sea un requisito fundamental para colgarnos la vitola de ingeniero, otras disciplinas también se hacen llamar así y tienen problemas en el desarrollo de sus proyectos, quizás el problema esta en que hemos querido emularlos en vez de darnos cuenta de las características tan peculiares que tiene el desarrollo del software. Poco a poco, parece que estamos creando una nueva tesis que sustituirá a la que en 1968 se impuso como modelo de desarrollo, esta bastante claro que los enfoques ágiles tienen muchas carencias (equipos de desarrollo pequeños, alta colaboración por parte del cliente, entornos ágiles, proyectos específicos con requisitos variable,....), pero nos han hecho darnos cuenta de que hay muchas maneras de desarrollar software, que cada proyecto necesita su propio enfoque y que quizás vitorear que utilizamos una metodología u otra, es un efecto comercial que puede no beneficiarnos tanto como parece. La selección de buenas prácticas, métodos y herramientas, que hayan sido probadas es sin duda una estrategia mucho más inteligente, disponer de un amplio abanico donde elegir nos facilitará mucho un entorno hostil y que parece que no desea que finalicemos nunca en la fecha o con todos los requisitos cubiertos.

Las últimas secciones de este capítulo han mostrado las principales características de seis metodologías ágiles, que tal y como se puede extraer de la selección previa, son las que gozan de mayor presencia en internet, disponen de mayor documentación y estudios y experiencias. El objetivo era identificar estas metodologías para que en el capítulo Comparativa de las metodologías, podamos realizar una comparativa que destaque sus diferencias, principales virtudes y adaptación a los proyectos empresariales con Java EE.

Referencias

[1] [Avison 1995] D. E. Avison and G. Fitzgerald, Information Systems Development: Methodologies, Techniques, and Tools,McGraw-Hill (1995).

[2] Fred Brooks. The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition. Ed. Addison Wesley, 1995. 2 edition.

[3] E, Georgiadou, "Software Proces And Product Improvement: A Historical Perspective". Cybernetics and Systems Analysis. Vol. 39, No. 1 2003.

[4] Jean-Claude Derniame, Badara Ali Kaba, David Graham Wastell (Eds.): Software Process: Principles, Methodology, Technology. Lecture Notes in Computer Science 1500 Springer 1999, ISBN 3-540-65516-6 BibTeX

[5]Pekka Abrahamsson, Outi salo & Jussi Ronkainen, Agile Software Development methods, review and analysis. VTT Electronics 2002.

[6] Managing the development of large software systems (context) - Royce - 1970

[7] The Standard Waterfall model for systems developments. http://web.archive.org/web/20040403211247/http://asd-www.larc.nasa.gov/barkstrom/public/The_Standard_Waterfall_Model_For_Systems_Development.htm

[8]Per Kroll, Bruce MacIsaac. Agility and Discipline Made Easy Practices From OpenUP and RUP. Ed. Addison Wesley, 2006

[9] History of Unified Process, Scott Amber, http://www.enterpriseunifiedprocess.com/essays/history.html

[10] Understanding RUP roles, http://www.ibm.com/developerworks/rational/library/apr05/crain/index.html

[11] Rational, the software development company. Rational Unified Process, best practices for software development teams. Rational Software WhitePaper, revision

[12] Case Studies for IBM Rational Software. http://tinyurl.com/rupcasestudies

[13] E.Mnkandla, B.Dwolatzky. Agile Methodologies Selection Toolbox.International COnference on Software Enfineering Advances (ICSEA 2007). IEEE 2007

[14] A.qumer, B.Henderson-Sellers. An Evaluation of the degree of agility in six agile methods and its applicability for method engineering. Information and Software Technology vol 50. 2008

[15] Martin Fowler. The New Methodology. http://martinfowler.com/articles/newMethodology.html

[16] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, Juhani Warsta. Agile software development methods, review and analysis. VTT Publications. ESPOO 2002.

[17] Pekka Abrahamsson, Juhani Warsta, Mikko T.Siponen, Jussi Ronkainen. New Directions on Agile Methods: A comparative Analysis. Procedings of the 25th International Conference on Software Engineering(ICSE'03). IEEE 2003.

[18] Jim Highsmith. A Baseline Agile Framework(DRAFT), 2008.
[19] Jim Highsmith. Agile Project Management: Creating Innovative Products.Addison-Wesley Professional (April 16, 2004)

[20] Foros de discusión, Yahoo Groups. www.groups.yahoo.com/group/agileprojectmanagement.

[21] Agile Project Leadership Network, http://apln.org/.

[22] Alistair Cockburn.Agile Software Development: The Cooperative Game. Addison Weasley Professional 2006. Second Edition.

[23] Alistair Cockburn. Surviving Object-Oriented Projects. Addison Weasley Professional. 1997.

[24] Stapleton. Dynamic Systems Development Method. The method in practice. Addison Weasley Professional. 1997

[25] Takeuchi, H y Nonaka, I. The New Product Development Game. Harvard Business Review Jan/Feb 1986.

[26] Juan Palacio, Flexibilidad con Scrum, principios de diseño e implantación en campos Scrum. SafeCreative. Edición Octubre 2007

[27] Henrik kniberg, XP and Scrum from the Trenches. How we do Scrum. InfoQ 2007.

[28] Schwaber, K y Beedle, M. Agile Software Development with Scrum. Upper Saddle River, Prentice Hall 2002.

[29] Rising, L y Janoff, N.S. The Scrum Software development process for small teams. IEEE Software 17(4) 2000.

[30] Kent Beck. Extreme Programming explained: Embrace Change. Reading, Mass. Addison Weasley 1999.

[31] Grenning, J. Launching Xp at a Process-Intensive Company. IEEE Software 18:3-9. 2001

[32] David Astels. Test-Driven Development. A practical guide. Prentice Hall PTR 2003.

[33] Christian Schmidkonz, CSM & Jürgen Staader, SAP AG, Germany. "Piloting of Test-driven Development in Combination with Scrum" http://www.scrumalliance.org/resources/267

[34] Patricio Leteiler et. al., An Experiment Working with RUP and XP, Extreme Programming and Agile Processes in Software Engineering, 4th International Conference, 2003.

[35] Martin Fowler, Continous Integration.2006 http://martinfowler.com/articles/continuousIntegration.html

[36] Interview with Hakan Erdogmus by Deborah Hartmann. Hakan Erdogmus on TDD Misunderstandings and Adoption Issues. 2008. http://www.infoq.com/interviews/TDD-Hakan-Erdogmus

6. Comparativa metodologías

En este capítulo nos proponemos presentar una comparativa de las metodologías ágiles seleccionadas en el capítulo cinco. Para ello hemos determinado un conjunto de criterios que consideramos relevantes y que nos ayuden a ver parámetros fundamentales de las metodologías, como son su estado actual, su adaptabilidad, fases que ocupan, que procesos describen, como miden la calidad, etc.

Contenidos de la sección

Criterios de la comparativa

Para la selección de los criterios que queremos utilizar en la comparativa de metodologías ágiles nos hemos apoyado en una comparativa que se realizó en el año 2002 [1][2] y que es el único estudio que hemos encontrado y que guarda alguna relación con nuestros objetivos. También hemos seleccionado diferentes criterios para la comparativa basándonos en los puntos primordiales que Larry Trussell propone como necesarios en toda metodología (ver sección Necesidades de una metodologías, capítulo 5), además de un conjunto de criterios que nos permitan enlazar las metodologías ágiles con las aplicaciones empresariales, realizadas con Java EE y presentadas en el capítulo 2. Finalmente hemos indicado las diferentes herramientas específicas que cada metodología, sin más ánimo que el de mostrar las diferentes herramientas que cada metodología propone.

En el capítulo cinco hemos realizado una caracterización de las diferentes metodologías ágiles, los puntos que utilizábamos para describir las metodologías eran:

- Procesos
- Roles y responsabilidades
- Practicas
- Adopción y experiencias
- Entorno de uso
- Estudios actuales

Estos puntos no han sido escogidos al azar, sino que conforman buena parte de lo que necesitaremos para realizar la comparativa y que ya fueron utilizados por Warsta y Ronkainen [1].

Los criterios seleccionados de estudios anteriores y que nos permiten observar puntos necesarios de una metodología han sido:

* Ciclo de vida del proyecto. Consideramos las siguientes etapas o fases:
  - Principio del proyecto.
  - Especificación de requisitos.
  - Diseño.
  - Codificación.
  - Pruebas unitarias.
  - Pruebas de integración.
  - Pruebas del sistema.
  - Pruebas de aceptación.
  - Sistema en uso o mantenimiento.
* Para cada una de las etapas del proyecto que ocupa especificamos:
  - Si da soporte para la gestión del proyecto en esa etapa.
  - Si identifica un proceso para utilizar junto al método en esa etapa.
  - El método describe prácticas, actividades y artefactos para esa etapa. Esto es equivalente ha decir que ofrece una guía concreta o no para esta etapa.
* Estado actual de la metodología:
  - Recién nacida.
  - En construcción.
  - Activa.
  - Olvidada.

- Guías prescriptivas vs guías adaptables.

Con estos criterios podemos observar cuantas fases de un proyecto cubre cada una de las metodologías, si provee una guía abstracta o por el contrario provee de prácticas y técnicas concretas, el estado de la metodología en función de su tiempo de vida, experiencias y adopciones observadas. Por último, observamos el grado de adaptabilidad de la metodología. Estos son alguno de los criterios que proponían en su estudio Ronkainen, Abrahamsson y Warsta [1], y que tenía como objetivo encontrar las similitudes y diferencias de las metodologías comparadas. De ellos podemos extraer puntos primordiales de una metodología como la utilización de artefactos como la visión del producto, utilización de un ciclo de vida, vinculación con el cliente, etc.
Un punto que consideramos primordial y que no se consideraba en la comparativa en la cual nos basamos es el tratamiento de los parámetros de calidad, por parte de las metodologías. De este modo consideramos necesarios establecer un criterio más para la comparativa y que cumple otro de los puntos necesarios en toda metodología:

- Calidad en la metodología.

Con tal de enlazar las aplicaciones empresariales Java EE y las metodologías ágiles seleccionadas, hemos optado por los siguientes criterios:

- Roles Java EE y Roles de las metodologías.
- Soluciones empresariales.

Finalmente hemos tenido en cuenta un último criterios, que ya hemos mencionado al principio de esta sección:

- Herramientas específicas de la metodología.

Estos criterios nos permiten evaluar varios de los parámetros que indica Larry Trussell y que indicábamos en el capítulo cinco, Necesidades de una metodología. Nuestro objetivo no es obtener que metodología es la mejor, ya que sería una comparativa poco útil además de estar conceptualmente equivocada. La intención es mostrar que diferencias existen entre las seis metodologías elegidas, que características tienen unas y cuales las otras y de este modo disponer de una visión crítica que nos ayude en la elección y aplicación de una metodología en un proyecto software concreto.
En la siguiente sección explicamos con mayor detalle cada uno de los criterios y realizamos la comparativa de las herramientas.

Comparativa

1. *Ciclo de vida del proyecto.*

En la figura 1 (ver página siguiente) mostramos cada una de las fases de un ciclo de vida estándar de un proyecto de desarrollo del software. Hemos indicado con un color intenso cada una de las fases y áreas que la metodología cubre y en un color pastel o difuminado, las fases que cubre la metodología pero no el área determinada Por ejemplo con la metodología Agile Project Management, se cubre la fase de Principio de Proyecto y lo hace dando indicaciones a nivel de gestión del proyecto, su modelo de procesos contempla esta fase y da indicaciones de como llevarla a cabo, y finalmente propone un conjunto de prácticas y artefactos a generar(caja del producto, listado de características), por tanto las tres áreas de esa fase estas coloreadas de un color intenso. En la fase diseño se provee de un modelo de gestión del proyecto, se contempla la fase en su proceso de manera explícita, pero no se da una guía concreta de prácticas a realizar, ni de artefactos a generar, por lo tanto las dos primeras áreas se colorean de un color intenso y la última difuminado. Las fases que no cumplen ningún área se deja de color blanco.
Un último ejemplo, si nos fijamos en la metodología Scrum, podemos observar que las fases de diseño, codificación y pruebas unitarias, tan sólo indican el área de gestión y ninguna de las otras dos, esto es porque Scrum da soporte de gestión y control de estas fases, pero en su modelo de procesos no las especifica, tan solo cuando estamos en la fase de pruebas de integración, ya que consideramos que se da lugar al final del sprint y que los artefactos requeridos son por ejemplo la demo del producto y las prácticas son las reuniones de fin de sprint, aceptación por parte del cliente del producto y decisión de si es necesario hacer más sprints y por tanto reiniciar el proceso.

De esta comparativa podemos extraer las siguientes conclusiones:

- Las metodologías ágiles están centradas en diferentes fases del ciclo de vida de desarrollo del software, algunas se enfocan a las fases de pruebas como puede ser TDD, otras al inicio y especificación de requisitos como APM, solo una cubre toda las fases, DSDM y la mayoría (todas menos TDD) contemplan el núcleo de las fases del ciclo de vida (Diseño, codificación y pruebas).

- Con respecto a las áreas estudiadas, algunas metodologías están más enfocadas a presentar diferentes prácticas como es el caso de Extreme Programming y TDD, otras a la gestión de proyectos, como Scrum, Agile Project Management, Crystal e incluso DSDM.

- Algunas metodologías son excesivamente abstractas, o esa es la impresión que nos deja el caso de Crystal y DSDM, que no establece ninguna práctica concreta o artefacto a realizar, si que puede exigir algunos artefactos, pero nunca establece como los quiere o como hacerlos.

- Por separado no parecen factible utilizarlas como metodologías únicas para un proyecto, necesitan alguna otra que las complemente o la adopción de practicas y procesos aislados complementarios.

- Crystal a pesar de cubrir muy pocas fases del ciclo de vida, exige la existencia y creación de artefactos en otras fases, como es la fase de especificación. Crystal requiere de un documento de especificación, preferiblemente que siga la técnica de especificación mediante casos de uso, pero no la incluye en la descripción de ninguna sus metodologías. Esta acción parece necesaria para determinar la criticidad y tamaño del proyecto, condición sine qua non de la elección de una de las metodologías de la familia Crystal y por tanto del inicio de la metodología.

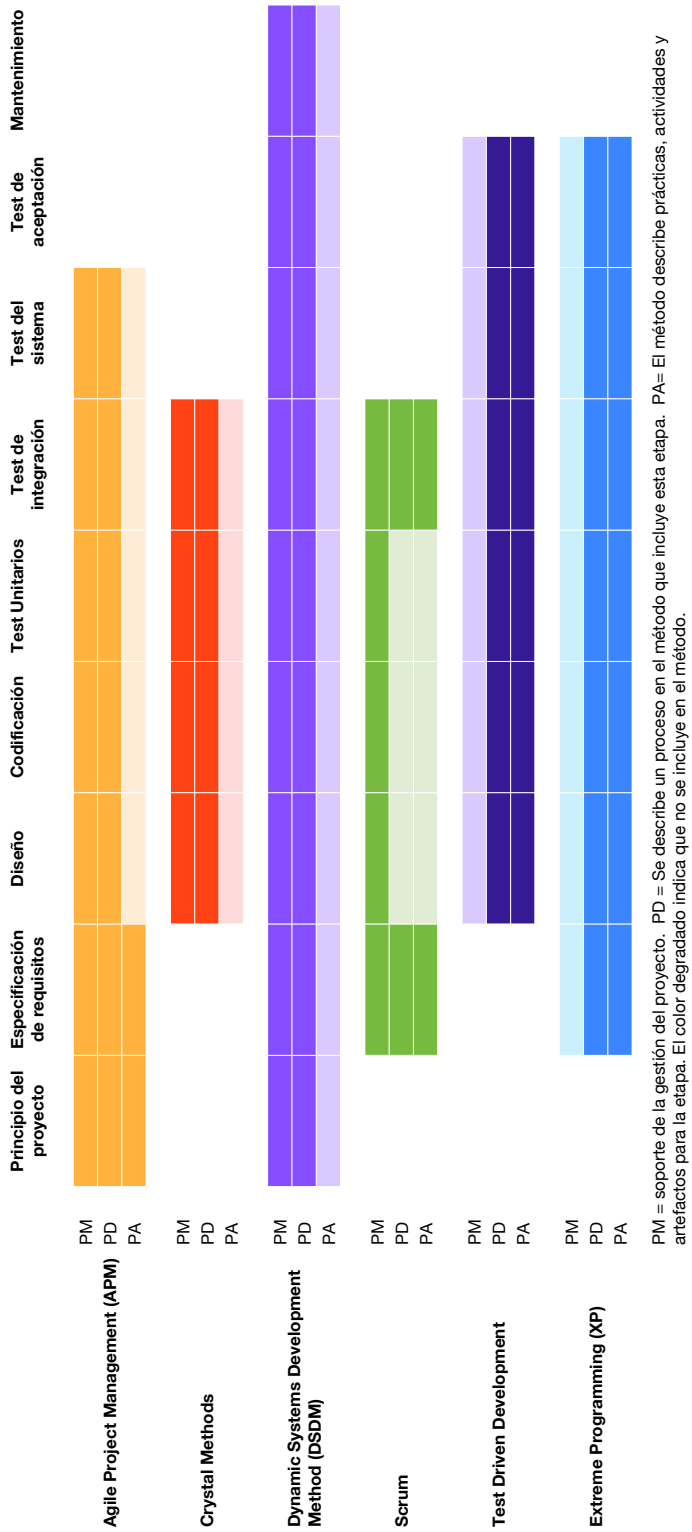

Principio del proyecto
Especificación de requisitos
Diseño
Codificación
Test Unitarios
Test de integración
Test del sistema
Test de aceptación
Mantenimiento
PM
PD
PA
Agile Project Management (APM)
Crystal Methods
Dynamic Systems Development Method (DSDM)
Scrum
Test Driven Development
Extreme Programming (XP)
PM = soporte de la gestión del proyecto. PD = Se describe un proceso en el método que incluye esta etapa. PA= El método describe prácticas, actividades y artefactos para la etapa. El color degradado indica que no se incluye en el método.

fig. 1

2. *Estado actual metodología.*

Los criterios para la selección de cada una de los estados de las metodologías están fundamentados en la caracterización que hemos realizado de las metodologías en el capítulo 5. De tal manera que una metodología en estado de:

- Recién nacida. Es aquella metodología que tiene un año o menos y de la cual no tenemos evidencias empíricas, ni estudios.
- En construcción. Aquellas metodologías con mas de una año de existencia, pero que no disponen de experiencias documentadas ni/o estudios.
- Activa. Son aquellas metodologías que llevan varios en el desarrollo del software y de las cuales podemos encontrar experiencias y estudios que corroboren su ratio efectividad.
- Olvidada. Aquellas metodologías que llevan el suficiente tiempo sin ser utilizadas y de las cuales no se encuentran experiencias actuales, ni estudios.

Metodología	Explicación	Estado 7/08
APM	A pesar de existir desde el año 2004, no ha mostrado la actividad académica o empresarial suficiente como para considerarse activa o ya establecida.	En construcción.
Crystal Methods	Activa con respecto a Orange y Clear, en construcción en cuanto a familia de metodologías incompleta.	En construcción/Activa
DSDM	Siendo una de las metodologías con mayor adopción en el Reino Unido y disponiendo de un gran número de experiencias se considera una metodología activa.	Activa
SCRUM	Una de las metodologías más antiguas y que más de moda esta últimamente, actualmente podemos encontrar muchos estudios y experiencias con Scrum.	Activa
TDD	La técnica/metodología más extendida. La podemos encontrar en un gran número de proyectos, muchas veces de la mano de XP.	Activa
XP	Desde 1999 se ha convertido en la punta del iceberg de las metodologías ágiles, es la primera en internet, en adopciones y experiencias y la primera en estudios.	Activa

tabla 1

Como principales conclusiones de esta comparativa podemos escoger el grado de madurez que están empezando a tener las metodologías ágiles, o al menos las que hemos escogido empieza a dar señales de que son una realidad palpable y que indican que un gran número de empresas están dando el salto y dejando de experimentar con ellas para utilizarlas seriamente.

3. *Guías prescriptivas vs guías ajustables.*

Parece incongruente hablar de metodologías ágiles y asociar el término prescriptivo, tan relacionado a las metodologías tradicionales. Cuando decimos que una metodología ofrece una guía prescriptiva, estamos declarando que las prácticas, métodos y procesos que enuncia la metodología son estrictas y deben seguirse tal y como publica el autor. De este modo, para decir que estas aplicando una metodología con guía prescriptiva, debes estar aplicando todos y cada uno de sus métodos, prácticas y procesos.
Este es un parámetro que puede decantar la balanza a hora de decidir la adopción de una metodología u otra, ya que implantar una nueva metodología en una empresa no es una tarea fácil. Es importante no sacar conclusiones precipitadas y considerar todas las implicaciones de una y otra aseveración, una guía prescriptiva asegura resultados a nivel de proyecto y es restrictiva al respecto, una guía no prescriptiva esta asegurando resultados a nivel individual y de proyecto, tan solo si se utiliza como si fuese prescriptiva. Obviamente la no prescriptiva es mas ajustable a diferentes tipos de proyectos, pero es necesario ser un "usuario avanzado" . Esto quiere decir que la metodología funciona, si, pero si se ejecuta completamente, las diferentes combinaciones que nosotros hagamos con otros métodos, corren a cuenta y riesgo nuestro, nadie puede asegurarnos que vaya a funcionar. Sin embargo, una metodología prescriptiva esta orientada a managers mas noveles, que deben limitarse (no siendo esto siempre sencillo) a seguir los pasos que otros ya han probado por nosotros, experimentado y finalmente recomendado.
Por lo tanto, ahora ya estamos preparados para entender las implicaciones de los resultados mostrados en la tabla 2.

Metodología	Guía prescriptiva
APM	No.
Crystal Methods	Sí.
DSDM	No.
SCRUM	No.
TDD	No.
XP	No.

tabla 2

La única metodología con un proceso prescriptivo es Crystal, esto es debido a que la selección del tipo de metodología (Crystal, Orange) se hace en función de el tamaño del proyecto, el grado de "criticidad" y la prioridad del mismo. Una vez escogida la metodología se deben seguir las reglas especificadas, el ajuste ya se ha hecho previamente al escoger la solución. A pesar de esta situación, Crystal no condiciona en exceso un proyecto del software, si realmente es adecuado para aplicar métodos ágiles, ya que es perfectamente compatible con otros métodos ágiles que ofrecen una guía más concreta, como puede ser FDD [3].
Por otro lado, las metodologías ágiles APM, DSDM, SCRUM, TDD o XP, son por definición ajustables, ya sea porque permiten un uso parcial o total de las mismas a través de sus prácticas (XP y TDD) o porque existen diferentes estudios que muestran su ajustabilidad con otras metodologías o su falta de guía prescriptiva, tal y como podemos ver en mayor detalle si observamos sus procesos y prácticas en el capítulo cinco.
Como conclusiones de esta comparativa podemos extraer que la adaptabilidad de las metodología ágiles a diferentes tipos de proyectos es un factor común de todas ellas o de la mayoría y que gracias a la falta de guías prescriptivas podemos experimentar total o parcialmente con los métodos y practicas que nos proponen.

4. *Calidad.*

En la tabla 3 podemos observar las diferentes metodologías ágiles del estudio, para cada una de las cuales indicamos de que maneras tratan la calidad de su producto, si es que lo hacen, en función de los parámetros que vimos en el capítulo 4.

Metodología	Calidad en la metodología
APM	• Conformidad a los requisitos. Mantiene la vista siempre en la visión y establece un rol responsable del producto. • Usabilidad mediante la eficiencia. Utilización de tarjetas de especificación de rendimiento. • Satisfacción cliente. Continuo feedback en las diferentes reuniones.
Crystal Methods	• Conformidad a los requisitos y consistencia y precisión (Fiabilidad), mediante pruebas de funcionalidad automáticas y regresivas. • Satisfacción del cliente, práctica dos usuarios revisan y valoran las versiones liberadas. • Usabilidad, cumpliendo el parámetro de claridad y precisión documentación, ya que exigen generación guía de usuario.
DSDM	En *RAD and quality principles*[4] muestran como DSDM trata los aspectos de calidad: • Conformidad a los requisitos, los documentos de especificación en DSDM deben tener criterios de calidad especificados, mediante las diferentes revisiones se comprueban. • La fiabilidad del producto mediante su atributo de consistencia y precisión, con la realización de pruebas dinámicas y prototipos.
SCRUM	• Conformidad a los requisitos. Mediante la utilización del rol de product owner y demos de producto. • Satisfacción del cliente, a través de las reuniones y demos presentadas.
TDD	• La fiabilidad a través de: 1. Consistencia y precisión. Escribir las pruebas primero y ser exhaustivas. 2. Robustez. Gracias a que disponemos de pruebas exhaustivas. 3. Trazabilidad. Cuando aparece una nueva funcionalidad, lo primero es escribir las pruebas para ella. 4. Simplicidad. Diferentes estudios [5] muestran que el código generado si se generan primero las pruebas contiene un mayor número de métodos, pero más simples. • La usabilidad a través de: 1. Fiabilidad, cuatro de los cinco parámetros de calidad se contemplan. 2. Habilidad de probar el software. Es obvia si escribimos código sobre pruebas ya implementadas. • La mantenibilidad a través de: 1. Modularidad. Nos referimos otra vez a diferentes estudios [5] que demuestran que el código generado con TDD tiende a disponer de un mayor número de métodos y ser más modular. 2. Legibilidad. A través de la técnica de refactoring. 3. Simplicidad. Mismos motivos arriba mencionados. 4. Estudios empíricos varios [7] • La adaptabilidad a través de la capacidad de modificar el software.

Metodología	Calidad en la metodología
XP	Fundamentalmente el análisis es el mismo que para TDD mas los siguientes parámetros: • Satisfacción del cliente. Los clientes son una pieza clave y son requeridos en diferentes fases de XP. • Conformidad a los requisitos. En el juego de planificación, los clientes escogen objetivos.

tabla 3

Con respecto a APM, su orientación a la calidad parece muy pobre, seria recomendable además de la utilización de técnicas y métodos concretos a partir de la fase de diseño, utilizar algún programa de Software Quality assurance[1] o modelos de procesos para la mejora de la calidad como CMMI.
Crystal orienta todo su control de calidad en los procesos de comunicación, si estos son buenos, derivan que obtendremos un buen resultado. Obviamente esto no asegura todos los parámetros de calidad que vimos en el capítulo cuatro. DSDM adolece de los problemas que tienen la mayoría de metodologías ágiles y es que su principal fuente de control de la calidad son los inputs de los usuarios y las pruebas que más o menos pueda llevar a cabo.
Scrum es una de las metodologías que con mayor asiduidad podemos ver asociada a un modelo de calidad de procesos, en especial CMMI [5] y por algo será. Scrum es una gran metodología de gestión de proyectos, pero no presenta prácticas concretas que aseguren la calidad, su propósito no es ese, sino tal y como se hace en rugby (haciendo el símil proyecto por balón), sacar el proyecto adelante de una forma colectiva y colaborativa.
Por el lado de TDD y Extreme Programming nos encontramos dos metodologías que obtienen unos resultados my satisfactorios en cuanto al tratamiento de la calidad, hechando de menos algunas técnicas de seguimiento de las mismas (gestión de la calidad).

Las conclusiones que podemos extraer de esta comparativa son que:

- Por regla general, las metodologías ágiles confían la calidad de sus productos a :
  - Feedback continuo con los clientes en cada iteración.
  - Bajar el indice de defectos en cada nueva iteración y mediante el uso intensivo de pruebas.
  - Extreme Programming y TDD, a pesar de no estar orientadas a la calidad, sino a la productividad, demuestran tener un conjunto de prácticas y métodos más eficientes a la hora de conseguir un producto de calidad.
- No podemos asegurar todos los parámetros que definíamos como intrínsecos de la calidad de un producto software, con tan solo las metodologías ágiles.
- Las técnicas de control de calidad no son los suficientemente concretas, son mas bien guías que necesitan complementarse con prácticas concretas.

5. *Roles Java EE y Roles de las metodologías.*

Esta comparativa muestra los roles de las aplicaciones Java EE que cubren las metodologías ágiles estudiadas.

[1] Medir la calidad de tu producto a lo largo de tus procesos, ver capítulo 4.

	Proveedor productos Java EE	Proveedor de herramientas Java EE	Proveedor de componentes de aplicación	Ensamblador de aplicación	Encargado de desplegar aplicación y administrador de sistema
APM	Proveedores.	Proveedores.	Equipo del proyecto.	Equipo del proyecto.	Equipo del proyecto.
Crystal Methods	No.	No.	Otros diseñadores-programadores	Facilitador técnico.	Probador
DSDM	No.	No.	Desarrolladores	Desarrolladores	Desarrolladores
SCRUM	No.	No.	Equipo de desarrollo.	Equipo de desarrollo.	Equipo de desarrollo.
TDD	No.	No.	Desarrollador	Desarrollador	Desarrollador
XP	No.	No.	Programador	Programador	Probador

tabla 4

Podemos observar en la tabla 4 que un modo más o menos general, las tareas descritas por cada uno de los roles específicos en un proyecto Java EE se cubren, a excepción del de proveedor. Tan sólo APM especifica el rol de proveedor y lo tiene en cuenta a nivel de riesgos y desarrollo del producto, por lo tanto podemos afirmar que tan solo APM soporta completamente los roles necesarios para el desarrollo de una aplicación empresarial con Java EE.

6. *Soluciones empresariales.*

Inicialmente se había planteado realizar una comparativa orientada a las soluciones empresariales presentadas en el capítulo tres y ese es el motivo de haber realizado una exploración y caracterización tan detallada de diferentes aplicaciones empresariales. A la hora de realizar esta comparativa nos hemos encontrado que nos faltaban criterios sólidos sobre los cuales fundamentar la elección de una metodología u otra y tampoco hemos encontrado experiencias previas que nos guiasen en este proceso.
Con tal de poder determinar que metodologías se adecuan mejor a cada tipo de solución y hacerlo de un modo científico y que tenga algún tipo de validez, requiere que trabajemos sobre soluciones muy concretas, ya que tal y como pudimos ver en el capítulo 3, dependiendo de las funcionalidades de las cuales dotemos a nuestras aplicaciones obtenemos diferentes escenarios, requisitos y en definitiva todo un ecosistema diferente. Por tanto, siguiendo el compromiso de realizar un trabajo serio y evitando obtener conclusiones erróneas, concluimos que si no trabajamos con soluciones muy concretas no tiene sentido llevar a cabo esta comparativa.
A pesar de no poder mostrar esta comparativa, si que podemos intuir cuales serían los resultados. Las metodologías que hemos seleccionado son lo suficientemente adaptables e independientes de las tecnología, como para ser utilizables con cada una de las soluciones empresariales enumeradas. Especialmente adecuadas para las soluciones de proceso de negocio personalizados e innovadores y existentes en el mercado pero no en la empresa.

7. *Herramientas.*

En la tabla 5 mostramos los principales grupos de herramientas específicos requeridos por cada metodología. No consideramos herramientas tan esenciales para el desarrollo del software como puede ser un IDE[2] (Eclipse, NetBeans,...) , herramientas ofimáticas (Word, Excel, kate, OpenOffice, ...).

[2] Entorno de desarrollo integrado.

Metodología	Herramientas específicas
APM	No especifica ninguna practica concreta que necesite de una herramienta específica.
Crystal Methods	Las únicas herramientas especificas mencionadas son una herramientas para el control de las versiones de código, una herramienta para la gestión y seguimiento de proyectos, una herramienta para medir el rendimiento y para pruebas, sin entrar en mas detalles.
DSDM	No especifica ninguna practica concreta que necesite de una herramienta específica.
SCRUM	A pesar de no ser necesaria ninguna herramienta especial, están surgiendo aplicaciones web que facilitan el seguimiento del proyecto y la generación de los distintos artefactos de la metodología, que frecuentemente se realizan con paquetes ofimáticos.
TDD	La mayoría de las herramientas especificadas están orientadas a la preparación y ejecución de pruebas. Tenemos las siguientes: • Herramientas para la realización de refactoring. Normalmente cualquier editor de texto sirve, actualmente diferentes IDEs ayudan en esta tarea indicando donde se invocan todos los métodos,atributos o clases modificados. • Herramientas de integración continua. Como es Hudson, Cruise Control • Herramientas de gestión de proyectos y compilaciones automáticas. Como es Maven. Ant es tan solo de compilación automática. • Repositorios de código. Como son CVS y Subversion. • Framework para la realización de pruebas unitarias. Especialmente conocido es el framework para desarrollo de pruebas unitarias JUnit. • Herramientas para medir el rendimiento de la aplicación, como es JMeter.
XP	Las herramientas especificadas para TDD.

tabla 5

Conclusiones

Al finalizar las seis comparativas de las herramientas seleccionadas extraemos las siguientes conclusiones:

- Las metodologías ágiles no contemplan todo el ciclo de vida tal y como lo hemos visto tradicionalmente, si queremos seguir todas y cada una de las fases de este necesitamos utilizar otras metodologías (RUP) o complementarlas, combinándolas entre ellas.

- El estado actual de las metodologías ágiles es activo y ganando cada vez más adeptos. Las metodologías que hemos seleccionado no podían sino indicarnos esto, ya que fue uno de los criterios de selección de las misma, pero como poco, podemos afirmar que cinco de las seis metodologías elegidas gozan de buena salud y que son una realidad del panorama del software actual.

- Las metodologías ágiles pueden presentar guías my concretas, con técnicas muy específicas, como es el caso de TDD y XP, o por el contrario mostrar guías abstractas, métodos que nos guían para realizar el mismo trabajo de una manera más eficiente (Crystal).

- Por regla general, las metodologías ágiles no presentan guías prescriptivas y son ajustables y adaptables una vez el proyecto se ha iniciado a situaciones cambiantes. Crystal es una excepción.

- Las metodologías ágiles están orientadas a la productividad y a sacar el proyecto hacia adelante, es por esto que cuando hemos comprobado cuantos parámetros de la calidad consideraban, nos hemos encontrado con que tan solo TDD y XP cubrían casi todo el espectro. Su modo de alcanzar la calidad es mediante la interacción continua con el usuario y la mejora incremental del producto en cada iteración. TDD y XP sin proponerse como objetivo la calidad consiguen de un modo excepcional cubrir la mayoría de los parámetros de la calidad, gracias a su técnica de implementar primero las pruebas.

- Los roles que presentan las diferentes metodologías ágiles no difieren en exceso, presentan siempre a un representante del cliente, normalmente responsable de verificar el cumplimiento de los requisitos, un experto en la metodología para guiar al equipo, un gestor del proyecto y el equipo de desarrolladores (con mayor o menor detalle). Destaca Agile Project Mangement por detectar el rol de proveedor y que es muy relevante en proyecto desarrollados con la tecnología Java EE. La repartición de los roles nos indica una adecuación completamente independiente de la tecnología y adaptable a la mayoría de ellas.

- Las herramientas requeridas por las metodologías ágiles son por regla general muy comunes, tan solo XP y TDD han introducido una revolución en este campo, con herramientas que implementen las técnicas tan concretas especialmente en las fases de pruebas. Destacamos las diferentes herramientas de gestión que están surgiendo para cada metodología y que permiten monitorizar sus diferentes indicadores e iteraciones.

Tal y como he indicado al inicio del capítulo, esta comparativa debería ayudarnos a decidirnos por una de las seis metodologías escogidas, en caso de necesitar una y en función del proyecto que vayamos a iniciar . Pero ahora nos damos cuenta de varias cosas que no tuvimos en cuenta:

- Podemos aplicar técnicas de una metodología, independientemente de esta.
- Podemos adaptar una metodología a nuestro proyecto.
- Podemos combinar las metodologías entre ellas.

Esta completamente fuera del estudio la consideración de cualquiera de estas alternativas.

Nos hemos encontrado con especiales dificultades a la hora de comparar la adaptación de las diferentes metodologías a los proyectos empresariales con Java EE. Si que hemos podido observar que todas las metodologías ágiles pueden ser utilizadas con mayor o menor éxito para la realización de cualquiera de las soluciones empresariales encontradas, pero la falta de información, que detectamos ya en el capítulo 3, ha resultado determinante. La variabilidad de una misma solución es lo suficientemente significativa como para hacernos muy complicada la tarea de selección u adopción de una metodología. Es por esto, que consideramos que la mejor manera de comparar la adopción de una o varias metodologías, en vez de otras, tan solo tiene sentido si establecemos un caso lo suficientemente concreto, invalidando cualquier tipo de generalización y por ende, de conclusión, sino tenemos un grupo de casos lo suficientemente relevante. este hecho no hace más que remarcar la característica que ya comentamos en el capítulo anterior, una metodología se hace necesaria para cada solución.

Al finalizar esta comparativa nos dimos cuenta de un hecho que una vez considerado resulta obvio, pero que inicialmente no hubiésemos sido capaces de vislumbrar. Si una solución concreta la podemos definir esencialmente a través de sus requisitos y tan solo a partir de soluciones concretas podemos observar la idoneidad de una metodología u otra, podemos concluir que la elección de una metodología de desarrollo debería realizarse siempre después de la especificación de los requisitos de la aplicación. Si retrocedemos hasta la primera comparativa que mostramos en este capítulo, podremos observar que esta es la propuesta que Alistair Cockburn presenta para su familia de metodologías Crystal y lo hace obligando a determinar parámetros como la criticidad o el tamaño del proyecto, que no se pueden conocer si no se ha realizado un estudio previo del mismo. Esta idea que resulta tan intuitiva nos permite ir aún más allá y enunciar un principio de incertidumbre para la elección de las metodologías. El principio de incertidumbre[3] que enuncia Watts S. Humphrey[4] para la determinación de requisitos nos indica que no podemos conocer todos los requisitos de un nuevo sistema hasta que hayamos finalizado el mismo y esté siendo usado por el cliente. Este principio es aplicable a la elección de una metodología, siempre y cuando consideremos válido el razonamiento de que una metodología es elegida con suficiente criterio, si y solo si conocemos todos los requisitos de la solución que queremos construir. De este modo, podríamos enunciar un nuevo principio de incertidumbre para las metodologías el cual diría algo así:

"Para un nuevo sistema software, no conoceremos la metodología que mejor se adecua al mismo hasta que el sistema no lo estén utilizando los usuarios"

Este principio nos ayuda a ver la importancia que tiene que una metodología sea capaz de adaptarse a lo largo del proceso de desarrollo y como la componente iterativa e incremental de su ciclo de vida, es un factor fundamental.

Mi opinión personal y que es fruto del estudio y realización de esta comparativa, es que las metodologías ágiles son un conjunto de prácticas y métodos que surgen de la experiencia y el estudio del desarrollo de muchos proyectos software. Si tienes mucha experiencia en este campo, seguramente ya hayas utilizado muchas de ellas (técnicas y métodos), con mayor o menor éxito y tengas tu propia opinión al respecto, sino, son una buena guía para conseguir buenos resultados en tus proyectos, la elección depende de tus necesidades. Si estas teniendo problemas de en la gestión de tus proyectos, prueba con Scrum o APM, si los problemas los tienes porque tenéis un ratio de errores en vuestro software muy elevado, utiliza TDD o XP.

---

3 El principio de incertidumbre dice, "Para un nuevo sistema software, los requisitos no serán completamente conocidos hasta que el usuario no lo haya usado."

4 Watts S. Humphrey fundó el programa de procesos del software del Instituto de Ingeniería del Software (SEI) en la Carnegie Mellon University. Desde 1959 hasta 1986 fue director de programación de IBM y actualmente forma parte del equipo de investigadores de la Carnegie mellon University.

Referencias

[1] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, Juhani Warsta. Agile software development methods, review and analysis. VTT Publications. ESPOO 2002.

[2] Pekka Abrahamsson, Juhani Warsta, Mikko T.Siponen, Jussi Ronkainen. New Directions on Agile Methods: A comparative Analysis. Procedings of the 25th International Conference on Software Engineering(ICSE'03). IEEE 2003.

[3] Agile Styles: Feature Driven Development and the Crystal Methodologies. Video http://www.infoq.com/presentations/fdd-crystal-agile-overview

[4] Paul Herzlich. RAD and quality principles. The Institution of Electrical Engineers. 1995

[5] Jeff Sutherland, Carsten Ruseng Jakobsen, Kent Johnson. Scrum and CMMI Level 5: The Magic Potion for Code Warriors. Proceedings of the 41st Hawaii International Conference on System Sciences - 2008

[6] David S. Janzen,Hossein Saiedian Does Test-Driven Development Really Improve Software Design Quality?IEEE Software 2008

[7] Raimund Moser, Marco Scotto, Alberto Sillitti, and Giancarlo Succi. Does XP Deliver Quality and Maintainable Code? G. Concas et al. (Eds.): XP 2007, LNCS 4536, pp. 105–114, 2007.

7. Herramientas

En este capítulo introducimos las herramientas de desarrollo del software como instrumentos necesarios para todo ingeniero y persona relacionada con el mundo del software. Presentamos una clasificación de las herramientas, a partir de la cual identificamos la mayoría de las herramientas necesarias en el desarrollo de una aplicación software. Finalmente concluimos el capítulo, relacionando las herramientas con las metodologías ágiles y las tecnologías Java EE.

Contenidos de la sección

Qué son las herramientas

Una herramienta es según la definición de la Real Academia de la lengua Española (RAE) , un instrumento del cual nos servimos para hacer algo. Por tanto, si aplicamos esta definición a las herramientas que utilizamos en el desarrollo de un proyecto software, lo que obtenemos es que una herramienta es un instrumento del cual nos servimos para construir software y que es utilizado en cualquiera de las fases de su desarrollo. Estos instrumentos generalmente son aplicaciones software, pero también pueden ser cualquier otro tipo de instrumento que nos ayude en el desarrollo de nuestro producto, como una calculadora o un simple bolígrafo, estas últimas no las tendremos en cuenta.
El concepto que realmente nos interesa en este capítulo es el de herramientas de ingeniería del software, ya que el desarrollo del software debe ser una actividad desarrollada por ingenieros que utilizan técnicas, métodos y herramientas adecuadas para cada solución. Una herramienta de ingeniería del software es un recurso que ayuda a las personas a realizar su trabajo de ingeniería del software. Para el propósito de este capítulo definiremos herramienta como: " Una aplicación software que se utiliza para desarrollar, probar, analizar o mantener otra aplicación software o su documentación"[1]. Esta definición la deberíamos extender de tal manera que la aplicación que denominamos herramienta pueda ser utilizada a lo largo de todo el proceso de desarrollo del producto o aplicación software final y por tanto, incluir tareas como la gestión de los procesos o detección de los requisitos. También consideraremos herramientas los plugins, frameworks, librerías y componentes, a pesar de no ser aplicaciones software en si mismas, pero si que son utilizadas por las mismas.

Necesidad de las herramientas

Las herramientas, como aplicaciones software que son, han ido evolucionando a lo largo de estos últimos años y adaptándose a las necesidades y tendencias de cada momento. Actualmente nos encontramos en una época dominada por las aplicaciones web y por metodologías de desarrollo que tienden a realizar iteraciones cortas, tal y como hemos podido ver en los capítulos anteriores que hacen las metodologías ágiles. Dentro de este nuevo entorno, necesitamos herramientas que se adapten a los procesos ágiles que se dan en el ciclo de vida del desarrollo del software, estas como hemos visto requieren muchas veces de actividades muy concretas y mecanismos sofisticados de integración.
Podemos observar como la popularidad del desarrollo del software de una forma distribuida o colaborativa es cada vez mayor, ya sea a través del outsourcing, comunicación a distancia o equipos de desarrollo virtuales[1] [3], incrementando la demanda de intercambio de información y trabajo apoyado en herramientas colaborativas.
No solo se han modificado los procesos de desarrollo del software, sino que los productos que desarrollamos ahora no son los mismos que desarrollábamos hace 15 años, tal y como hemos comentado al principio de esta sección, actualmente las aplicaciones web ocupan la mayoría de los recursos de las consultorías informáticas. Pero no es tan solo eso, sino que las aplicaciones actuales cada vez son más grandes, complejas y diversas, utilizan nuevas tecnologías, componentes y métodos, que obviamente necesitan herramientas nuevas y específicas. Los desarrolladores tienen nuevas necesidades, ahora requieren de mejores herramientas para especificar, diseñar, generar, probar y desplegar complejos sistemas distribuidos e integrar diferentes tecnologías de sistemas heredados. Necesitan soporte para el diseño de interfaces de usuario cada vez más exigentes, control de versionado, gestión de la configuración, documentación y reutilización de un gran número de artefactos, ... En definitiva, los procesos y productos generados son más complejos que hace años y es necesaria la utilización de herramientas que nos faciliten la realización de los proyectos mejor, más rápidamente y de un modo más eficiente.

1 Un equipo de desarrollo virtual esta formado por personas físicas pero que no se encuentran físicamente en la misma oficina. Normalmente son equipos de desarrollo que se comunican y trabajan a través de la red.

Clasificación de las herramientas

Existen diferentes tipos de clasificaciones o categorías por las cuales podemos identificar las herramientas de desarrollo de nuestros proyectos de software. Con tal de determinar esta clasificación hemos utilizado un directorio de herramientas de desarrollo del software que podemos visitar en internet, softdevtools[5] y el cual dispone de hasta 900 herramientas comerciales y de libre distribución, también hemos consultado una comunidad de desarrollo que da soporte a todo tipo de proyectos de desarrollo de herramientas open source [6], wikipedia[7] y algunos estudios que hemos encontrado al respecto[2][4].
Si nos fijamos en el repositorio de herramientas que hemos mencionado antes [5], nos propone una clasificación muy intuitiva y que esta orientada a clasificar las herramientas de diferentes formas, estas son las categorías que establece:

- Licencia. Según el tipo de licencia con la que se distribuye, puede ser comercial, ope-source, freeware, shareware.
- Lenguaje. Según el lenguaje de programación en el cual ha sido desarrollada. Por ejemplo Java, .Net, Cobol,etc.
- Plataforma. Plataforma en la cual se puede ejecutar la aplicación, windows, unix, linux, web, mainframe, etc.
- Forma de la herramienta. Aquí especifica si la herramienta es una aplicación software o por el contrario un componente, librería, framework o plugin.
- Gestión de proyectos. Se incluyen todas las herramientas que estiman, planifican y permiten hacer un seguimiento del proyecto.
- Gestión de la configuración. Incluye herramientas de control de versiones, comparación de código, despliegue y compilación automáticas.
- Pruebas. Incluye las herramientas necesarias para el desarrollo de las pruebas.
- Enfoque. Según el enfoque de desarrollo, si se utilizan metodologías ágiles, CMM, orientado a objetos (UML), etc.
- Gestión de los datos. Especifica herramientas de gestión de los datos, como pueden ser las bases de datos.

Esta clasificación es redundante y no nos permite observar el ámbito de aplicación de cada herramienta, establece algunas características interesantes que bien podrían simplemente indicarse como atributos de la herramienta, en algunos aspectos es también demasiado específica. Esta fuente nos ha resultado muy útil para encontrar gran variedad de herramientas.
La clasificación que esta presente en wikipedia es muy simple y ni tan siquiera la vamos a mostrar aquí, preferimos hablar de la otra clasificación que establece tigris [6] y que es la siguiente:

- Análisis. Herramientas para la realización de análisis automático del código, métricas y otros parámetros.
- Construcción. Herramientas para codificar, probar y debugar.
- Diseño. Herramientas de diseño de software.

- Seguimiento de problemas. Herramientas de seguimiento de fallos, errores y bugs.
- Librerías. Aquí se ubican componentes software reusables.
- Procesos. Esta categoría contiene proyectos en los cuales se desarrollan herramientas relacionadas con el tratamiento de los procesos de desarrollo del software.
- Requisitos. Podemos encontrar herramientas para la gestión y tratamiento de los requisitos.
- SCM. Son las siglas de Software Configuration Management o lo que es lo mismo, gestión de la configuración del software y más comúnmente conocido como control de versiones del código y las herramientas más conocidas son CVS y SVN. Esta categoría incluye herramientas SCM y utilidades relacionadas.
- Comunicación "tecnológica". Categoría que engloba las herramientas que permiten la comunicación de los diferentes miembros del equipo de desarrollo.
- Pruebas. Herramientas para la realización de pruebas automatizadas.

Esta clasificación esta más próxima de los objetivos que buscamos, ya que podemos observar que las categorías, que utiliza en la clasificación de las herramientas, identifican varias de las tareas que se llevan a cabo en el desarrollo del software. De todos modos y al igual que pasa con la clasificación que hemos mostrado de softdevtools[5], son clasificaciones que se han hecho a posteriori y con un conjunto de herramientas definido, es decir que su objetivo no es englobar el mayor número de herramientas, sino clasificar las herramientas de las cuales disponen. A nosotros nos gustaría poder elaborar una clasificación en la cual veamos de un modo intuitivo y natural las herramientas necesarias para el desarrollo de un proyecto software.
Los dos estudios que presentamos a continuación nos resultan mucho más útiles en nuestra tarea de identificación y clasificación de las herramientas de desarrollo del software. En el reporte técnico del SEI[4] que data de 1987 se nos indica una metodología para clasificar herramientas de ingeniería del software. Proponen los siguientes campos a la hora de identificar una herramienta:

- Nombre de la herramienta.
- Número de la versión/ Fecha de liberación.
- Distribuidor de la herramienta, incluyendo dirección y número de teléfono si dispone. Actualmente deberíamos incluir dirección de correo electrónico.
- Pequeña descripción de una o dos líneas.
- Fases de desarrollo. Este es el punto más interesante que proponen y consiste en indicar para cada herramienta en que fases de desarrollo se ve involucrada y que operación lleva a cabo. Para cada una de las fases puedes seleccionar las operaciones de crear, transformar, agrupar, analizar, refinar, importar, exportar u otros (ver anexo). Y estas son las fases consideradas
    - Gestión del proyecto.
    - Sistema/Software análisis de los requisitos.
    - Fase de diseño preliminar.
    - Fase de diseño detallado.
    - Codificación y test unitarios.
    - Integración de componentes y fase de pruebas.
    - Fase de configuración y pruebas.
    - Fase de integración del sistema y pruebas.
- Definiciones especiales.
- Objetos y operaciones.
- Atributos.

- Comentarios.

Para poder ver con mayor detalle la técnica propuesta por el SEI ver el anexo tres, donde incluimos estas indicaciones y un ejemplo de uso en inglés, extraído del mismo reporte.
Esta última propuesta de clasificación da una mayor relevancia a las fases de desarrollo de un proyecto y tal y como veremos en el estudio que presenta Grundy [2], es una de las mejores maneras de identificar y presentar las herramientas necesarias en el desarrollo de una aplicación software. Grundy y Hosking realizaron una clasificación de las herramientas necesarias para el desarrollo del software y lo hicieron simplificando las fases de desarrollo que podemos encontrarnos en todo proyecto. Escogieron las cuatro más esenciales: requisitos, diseño, implementación y pruebas. Para cada una de estas fases mostraron las siguientes categorías de herramientas (ver figura 1):

- Herramientas de soporte a los procesos como :
  - Herramientas de procesos centrados en entornos de ingeniería de software (PCSEEs).
  - Sistemas de gestión de flujo de trabajo (WFMS).
  - Herramientas de gestión de proyectos y estimación de costes.

- Herramientas de ingeniería de requisitos. Ya sean formales o con lenguaje natural.
- Herramientas de análisis y diseño, muchas de estas herramientas se engloban dentro de las famosas herramientas CASE (Computer-Aided Software Engineering).
- Herramientas para el diseño y la arquitectura del software.
- Herramientas 4GL (cuarta generación de lenguajes) o generadoras de aplicaciones. Incluyen herramientas visuales de programación y herramientas de diseño de interfaces de usuario.
- Herramientas de programación, que incluyen los IDEs o entornos integrados de programación.
- Herramientas de debug, pruebas, monitorización y otras herramientas de evaluación.
- Herramientas de control de versionado y gestión de la configuración.
- Herramientas para la documentación.
- Herramientas para trabajo colaborativo.

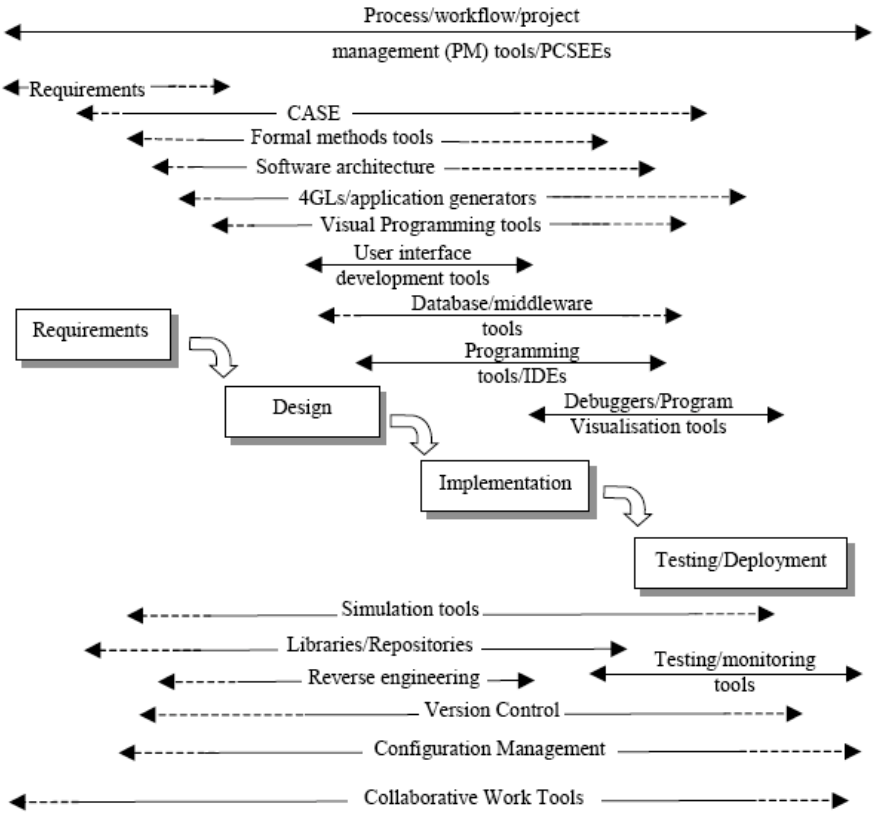


fig. 1

Esta es para mi la clasificación y la división en categorías más completa que he encontrado hasta la fecha.

Las categorías que explicaré en las siguientes secciones han sido extraídas de este último estudio, incluyendo o detallando aquellas que hemos considerado más relevantes o que necesitaban aclararse.

Herramientas de gestión de proyectos y estimación de costes

No me gusta incluirlas tal y como lo hacen Grundy y Hoskin[4] como herramientas que dan soporte a los procesos, ya que tanto la estimación de costes como la gestión de proyectos contienen una componente humana muy importante y restringirla a tan sólo los procesos es obviar este factor tan determinante.

1. *Fases del ciclo de vida.*

Las herramientas de gestión de proyectos y estimación de costes se utilizan a lo largo de todo el proyecto, ya que deben realizar el seguimiento y control del mismo, ir actualizándose y modificándose según la evolución del mismo.

2. *Funcionalidad.*

Son herramientas que nos permiten realizar tareas de planificación y gestión de los equipos a lo largo del proyecto, así como el cálculo de esfuerzos, costes y otras variables significativas que deban controlarse a lo largo del proyecto.

3. *Ejemplos.*

Podemos encontrarnos herramientas de propósito general o herramientas más especificas como MS Project, Rational Clear Guide, Platinum Process Continuum o Foresight. Por nombrar alguna herramienta específica de metodologías ágiles contamos con IceScrum 2 o Agile Planner.

Herramientas de soporte a los procesos

Consideramos herramientas que dan soporte a los procesos a las herramientas de procesos centrados en entornos de ingeniería de software (PCSEEs) y a los sistemas de gestión de flujo de trabajo (WFMS).

1. *Fases del ciclo de vida.*

Al trabajar con los procesos de desarrollo y procesos que se ejecutan a lo largo del proyecto, hemos de utilizarlos a lo largo de todo el proyecto y por tanto están ligadas a todas las fases del ciclo de vida de desarrollo de un producto software.

2. *Funcionalidad.*

Estas herramientas permiten a los desarrolladores caracterizar los procesos que ellos usan, ejecutarlos, y coordinar el desarrollo del software utilizando los procesos. Los PCSEEs o herramientas de procesos centradas en entornos de ingeniería del software, normalmente están centradas en el desarrollo del software, mientras que los WFMS o sistemas de gestión de flujo de trabajo son herramientas que se centran en procesos de propósitos más general, cubriendo un espectro de procesos mas grande.

3. *Ejemplos.*

Algunos ejemplos de herramientas de soporte de procesos son ProcessWeaver, SPADE o MILANO.

Herramientas de ingeniería de requisitos

Mantenemos esta categoría a pesar de englobar un número muy grande de herramientas, genéricas y específicas, formales y de lenguaje natural. La consideramos una categoría debido a que cada día gana mayor importancia el proceso de captura y especificación de requisitos y porque es importante destacar que existen herramientas específicas para esta tarea, aunque la mayoría engloben estos procesos y otros como el diseño y la implementación de la aplicación.

1. *Fases del ciclo de vida.*

Para ser estrictos tan solo deberían utilizarse las herramientas de ingeniería de requisitos o las funcionalidades correspondientes de la herramienta, en las fases de recogida de requisitos y especificación de la solución software. Esta situación se da tan solo una vez en proyectos gestionados de una forma tradicional y varias veces en proyectos ágiles. Pero una herramienta realmente útil es aquella que nos permite un seguimiento de los requisitos (trazabilidad) a lo largo de todo el proyecto y su posterior validación, por lo que la herramienta se puede utilizar a lo largo del proyecto para realizar el seguimiento de las funcionalidades/requisitos implementados y los que han sido validados por el cliente (fases de implementación y pruebas de aceptación).

2. *Funcionalidad.*

Las herramientas de ingeniería de requisitos soportan la obtención, codificación, validación, y evolución de los requisitos de usuarios y sistemas. Suelen incluir soporte tanto para la captura como el análisis de los requisitos funcionales y los no funcionales.

3. *Ejemplos.*

Existen diferentes herramientas que nos permiten la descripción de los requisitos en lenguaje natural como IDIOM o Rational Requisite Pro y otras que utilizan técnicas formales de especificación como RATS o PARSEDAT.

Herramientas de ingeniería del software asistidas por ordenador

Más conocidas con el nombre de Case Tools, dan soporte a prácticamente todas las técnicas existentes de especificación de sistemas software, métodos de análisis, diseño e incluso codificación. Podrían incluirse como herramientas de ingeniería de requisitos, pero debido a que cubren un rango tan amplio de fases y tareas hemos decidido mantenerlas como una categoría independiente.

1. *Fases del ciclo de vida.*

Las herramientas CASE pueden considerarse que se utilizan o que pueden ser utilizadas en todas las fases del ciclo de vida de una aplicación software, exceptuando la fase de pruebas y mantenimiento. A veces son utilizadas como únicas herramientas de soporte en las fases que ocupan, pero normalmente podemos encontrarnos con herramientas más específicas que las complementan, ya se a la hora de especificar los requisitos, modelar el dominio de la aplicación o diseñarla.

2. *Funcionalidad.*

Estas herramientas soportan técnicas de especificación de sistemas basadas en objetos, como el análisis y diseño orientado a objetos, basadas en funciones como los diagramas de flujos de datos, diagramas de secuencia de mensajes gráficos (MSCs) y diagramas de transición de estados, orientadas a los datos, como diagramas de entidades relacionales (ERDs). También dan soporte a tareas de diseño, modelado, análisis y refinamiento de las decisiones de implementación. Para ofrecer estas técnicas nos brindan la posibilidad de realizar diagramas completos, generar código, trabajar con el lenguaje unificado de modelado UML y otras utilidades que necesitemos para llevar a cabo las técnicas mencionadas.

3. *Ejemplos.*

Mencionamos las herramientas CASE más representativas Rational Rose y Platinum ERWin.

Herramientas de diseño y arquitectura del software

Otros subconjunto de funcionalidades que incluyen las herramientas CASE, son las funcionalidades de las cuales se ocupan las herramientas de diseño y arquitectura del software o del sistema.

1. *Fases del ciclo de vida.*

Estas herramientas son utilizadas en las fases de especificación, diseño e implementación de la aplicación. Esto es debido a que se encargan de tareas que se realizan en estas fases del desarrollo de toda aplicación software.

2. *Funcionalidad.*

Estas herramientas son utilizadas para modelar , analizar y afinar las decisiones necesarias a la hora de realizar la implementación de la aplicación. Con ellas se especifica la arquitectura del sistema, se diseña y también se puede generar código automáticamente o realizar ingeniería inversa, ocupando las tres fases del ciclo de vida que hemos identificado.

3. *Ejemplos.*

Algunos ejemplos de este tipo de herramientas son Clockworks o Dali.

Herramientas 4GL

Las herramientas generadoras de aplicaciones o mas comúnmente conocidas como herramientas para lenguajes de cuarta generación proveen de una mezcla de funcionalidades entre el diseño y la implementación.

1. *Fases del ciclo de vida.*

Esta categoría de herramientas soporta completamente las fases de diseño e implementación o codificación de la aplicación, tal y como hemos podido intuir de la breve descripción con las que las hemos presentado.

2. *Funcionalidad.*

La mayoría de este tipo de herramientas están centradas en facilitar el diseño asociándose a un lenguaje de programación concreto, tecnología y arquitectura, permitiéndonos realizar tareas como la creación de las tablas de una base de datos, formularios o reportes. Identificamos las herramientas de diseño de interfaces de usuario como herramientas 4GL, las cuales nos permiten diseñar complejas interfaces de usuario y reportes estructurados.

3. *Ejemplos.*

En función de las tareas que queramos llevar a cabo nos encontramos con diferentes herramientas, por ejemplo contamos con MS Access para las tareas relacionadas con las bases de datos y Garnet o Borland Delphi como herramientas de diseño de interfaces de usuario.

Herramientas de programación.

Quizás una de las categorías que mas y mejor ha evolucionado de entre todas las que estamos identificando. Inicialmente se utilizaban de forma aislada y ya fuesen como editores de textos con algunas funcionalidades añadidas, compiladores, ensambladores o debuggers y han ido evolucionando dando lugar a complejos entornos integrados de desarrollo (IDEs).

1. *Fases del ciclo de vida.*

Si somos completamente estrictos, la fase en la cual son utilizados el ciento por ciento de las herramientas de programación es en la fase de implementación o codificación de la aplicación. Debido a la evolución tan importante que han sufrido, los IDEs incluyen funcionalidades que nos permiten realizar tareas que van desde el diseño de las misma aplicación hasta la ejecución de las pruebas, actualización de las versiones del repositorio, etc. Por tanto podemos asegurar que los IDEs se utilizan en las fases de diseño, implementación y pruebas en el desarrollo de la mayoría de las aplicaciones,

aunque las herramientas propias de programación tan sólo en la fase de implementación.

2. *Funcionalidad.*

Las herramientas de programación se centran principalmente en proveer facilidades a la hora de llevar a cabo la implementación de la aplicación, están especializadas en un único lenguaje o a una familia de lenguajes acotada y pueden incorporar generadores de documentación, diseñadores de interfaces de usuario, navegadores internos e incluso debuggers.
No hemos mencionado las herramientas de programación visual en la introducción de esta categoría, pese a ser un conjunto de herramientas de programación muy importante. Estas herramientas enfatizan el uso de representaciones gráficas para la creación de aplicaciones, es decir, a través de editores visuales facilitan la creación de aplicaciones.

3. *Ejemplos.*

Existen muchas herramientas que integran editores, compiladores, debugger y facilidades a la hora de implementar, muchas de ellas incluyen ya funcionalidades para utilizar repositorios de código con control de versionado y otras herramientas colaborativas, quizás las más famosas son Eclipse, NetBeans, JBuilder, Delphi y Visual Studio 2005.

Herramientas para pruebas.

Una de las medidas más utilizadas para asegurar la calidad del software es mediante la ejecución de pruebas y la depuración (debugar) del código. Consideramos herramientas de pruebas todas aquellas que nos permitan probar las funcionalidades de la aplicación, su comportamiento, detectar sus errores y en definitiva indicarnos la calidad del producto, así como aquellas que nos permiten visualizar el comportamiento de nuestro algoritmos facilitándonos la corrección de posibles errores y medir los diferentes parámetros indicativos de la calidad del producto software desarrollado.

1. *Fases del ciclo de vida.*

La fase en la cual se llevan a cabo las pruebas del sistema es en la fase de pruebas, pero también ejecutamos pruebas unitarias y depuramos el código en la fase de implementación. Las herramientas necesarias para obtener las diferentes métricas de calidad del producto se ejecutan a lo largo de la fase de implementación pruebas

2. *Funcionalidad.*

Las herramientas para depurar código suelen centrarse en el análisis del código a muy bajo nivel y nos permitiéndonos una ejecución del código de la aplicación por pasos y mediante “breakpoints” o puntos de partida, mostrándonos información relevante de los datos, memoria y procesos lanzados. Normalmente esta herramienta viene integrada en la mayoría de entornos integrados de programación (IDEs).
Principalmente se llevan a cabo tres tipos de pruebas, pruebas unitarias, pruebas de aceptación y pruebas de integración. Existen herramientas para generar y ejecutar pruebas unitarias y casos de prueba de integración, otras herramientas se encargan de recoger los resultados e interpretarlos. Las herramientas de integración continua nos permiten ejecutar pruebas unitarias, compilar el código y siempre a través de la última versión del código que se encuentra en un repositorio, guardando y mostrándonos estadísticas de éxito y de errores y permitiéndonos programar cuando queremos realizar estas tareas.
Las herramientas que miden la calidad del producto utilizan los reportes generados por las herramientas que ejecutan las pruebas, evalúan el estilo del código, etc.

3. *Ejemplos.*

Existen herramientas específicas para depurar código, como GDB o DBX, pero por regla general siempre las veremos integradas en IDE's. Algunos ejemplos de herramientas de visualización son Tango o Balsa. Y finalmente, ejemplos de herramientas que ejecuten pruebas unitarias y casos de integración es el Rational Team Test, como herramienta de integración continua mencionamos Hudson y CruiseControl. Una herramienta específica que se utiliza para verificar el estilo del código es CheckStyle, otra herramienta que se utiliza para determinar métricas de calidad es FindBugs y Jupiter.

Herramientas control de versiones y gestión de la configuración.

Este tipo de herramientas han ido ganando cada vez más adeptos y han llegado a convertirse en unas herramientas fundamentales para todo desarrollo de software que se precie. Permiten gestionar y controlar la evolución de los artefactos que se generan a lo largo de todo el proyecto, guardando diferentes versiones de la mismas.

1. *Fases del ciclo de vida.*

Tanto las herramientas de control de versiones, como las de gestión de la configuración, son herramientas que se utilizan a lo largo de todo el proyecto, ya que ambas trabajan con la evolución del proyecto. Las herramientas de control de versiones nos permiten acceder a versiones anteriores de nuestro artefactos y las herramientas de gestión de la configuración nos permiten seguir los cambios hechos en una parte del sistema.

2. *Funcionalidad.*

Tal y como ya hemos introducido, las herramientas de control de versiones lo que hacen es ir guardando diferentes versiones de nuestros artefactos, normalmente será de nuestro código, pero también se utiliza para los documentos de especificación, diseño, etc. La utilización de estas herramientas nos permite revertir casi cualquier decisión errónea y garantizar un sistema más fiable, a la vez que facilitan el trabajo colaborativo, donde diferentes equipos pueden estar subiendo y bajando diferentes partes de la aplicación, disponiendo siempre de la versión más actualizada.
Las herramientas de gestión de configuración proveen soporte para la compilación de las versiones del repositorio seleccionadas e incluyen las herramientas de integración continua, de las cuales ya hemos hablado en la categoría anterior. También incluimos dentro de esta categoría las herramientas de compilación automática, que van íntimamente relacionadas con la integración continua, la ejecución de pruebas y las más actuales con la gestión del ciclo de de vida de la aplicación. Relacionada con las herramientas de gestión de la configuración están las herramientas de gestión de los cambios, las cuales se encargan de realizar un seguimiento de los cambios que se realizan a lo largo de todo el ciclo de vida de la aplicación. No podemos olvidar herramientas tan necesarias como las de seguimiento de errores o "bugtracking" que permiten mantener un listado priorizado de fallos que deben corregirse, ademas de un sin fin de utilidades colaborativas, necesarias para la realización de las modificaciones necesarias.

3. *Ejemplos.*

Las herramientas más conocidas y utilizadas para el control de versiones y como repositorios de código, son SVN, CVS y MS SourceSafe. Herramientas de gestión de la configuración son Rational ClearCase o Continous Change Manager Suite. Como herramientas de bugtracking mencionamos Jira o Bugzilla y como herramientas de compilación automática, Ant y Maven.

Herramientas para documentar.

Una tarea frecuentemente olvidada y que resulta fundamental cuando nos encontramos en proyectos de integración, adaptación, ampliación o modificación de sistemas existentes, es la documentación. Cada vez esta tarea esta mejor soportada y se nos ofrecen diferentes herramientas que la facilitan.

1. Fases del ciclo de vida.

La documentación de un proyecto empieza en el primer instante que se inicia el proyecto, ya sea con un documento de visión y alcance, o con una descripción de los requisitos por parte del cliente y finaliza con el mismo proyecto. Por tanto, las herramientas necesarias para documentar son utilizadas a lo largo de todas las fases de desarrollo de un proyecto software.

2. Funcionalidad.

Su principal función es la de facilitar la tarea de documentación a partir de las especificaciones del sistema y el código. Podemos considerar que las herramientas CASE dan soporte a la documentación, ya que nos permiten elaborar diseños y especificaciones visuales. También nos encontramos herramientas que se encuentran incluidas dentro de los IDEs y que dan soporte a la documentación del código.

3. Ejemplos.

Podemos nombrar prácticamente todas las herramientas incluidas en los paquete ofimáticos y también podemos refererirnos a herramientas como Javadoc que nos facilitan la documentación del código a través de su integración en un IDE.

Herramientas para realizar trabajos colaborativos.

Las aplicaciones software cada vez se desenvuelven en ambientes más competitivos y en entornos de trabajo distribuidos, donde la colaboración juega un papel muy importante a la hora de realizar un proyecto con éxito.

1. Fases del ciclo de vida.

Las herramientas que podemos identificar, que nos sirven para realizar nuestro trabajo diario en colaboración con nuestros compañeros, son utilizadas a lo largo de todas las fases del proyecto. No podemos restringir su uso a una fase determinada, ya que el intercambio de información debe ser fluido a lo largo de todo el proceso de desarrollo, sin excepción.

2. Funcionalidad.

Las herramientas colaborativas se caracterizan por permitir a un grupo de trabajo desarrollar sus tareas como un equipo distribuido, tanto en el tiempo como en el espacio. Podemos considerar prácticamente todas las actividades necesarias para mantener una comunicación fluida y todos los recursos de los que dispongamos, para no notar las diferencias horarias y sobretodo las diferente ubicaciones en la que se puede estar trabajando. De este modo, como herramientas colaborativas nos podemos encontrar desde simples sistemas de mensajería, pasando por clientes de correo, calendarios compartidos, hasta sistemas más complejos que permiten la compartición de documentos, noticias, comentarios y otros con funcionalidades atractivas como pizarras compartidas, directorios remotos, videoconferencia, etc.
Tal y como hemos mencionado en las herramientas de gestión de la configuración, las herramientas de seguimiento de fallos también incluyen un conjunto de funcionalidades colaborativas, ya que las

modificaciones del producto implican la colaboración y comunicación de diferentes componentes del equipo.

3. *Ejemplos.*

Prácticamente todos hemos utilizado herramientas colaborativas, aunque quizás nunca nos hayamos parado a pensarlo, este es el caso de los servicios de mensajería instantánea o chats. Otras herramientas que podemos encontrar en el mercado son Lotus Notes, MS Outlook, MS NetMeeting o BSCW.

Herramientas para metodologías ágiles y aplicaciones Java EE

Hemos querido incluir esta sección dentro de este capítulo con tal de enlazar el contenido de todo el documento y mostrar la tendencia actual con respecto a las herramientas, las metodologías ágiles y las aplicaciones Java EE.

Las metodologías ágiles nos proponen un conjunto de técnicas y métodos que varían el flujo tradicional del ciclo de vida de un proyecto software, a la vez que introducen conceptos novedosos que requieren de herramientas específicas. En el capítulo siete mostramos las herramientas específicas necesarias para las metodologías ágiles que comparamos y en resumen podríamos destacar la tendencia a la aparición de herramientas de gestión de proyectos ágiles, como AgilePlanner o IceScrum 2, que permiten las gestión de los procesos iterativos que se llevan a cabo en las metodologías ágiles. También juega un papel muy importante la metodologías Extreme Programming, ya que realiza un uso extensivo de las herramientas necesarias en las fases de pruebas, introduciendo el concepto de servidor de integración continua y la utilización de repositorios de código y herramientas de compilación automáticas.

Las aplicaciones Java EE disponen sin lugar a dudas de una gran variedad de herramientas, situándose como una de las tecnologías con un mayor número sin lugar a dudas. Si nos dirigimos a una fuente concreta [8] para chequear cuales son las categorías mencionadas anteriormente, que podemos encontrar en el desarrollo de una aplicación Java EE, nos encontramos con las siguientes :

- Herramientas de compilación automáticas. Hablamos de estas herramientas cuando especificamos la categoría herramientas de control de versiones y gestión de la configuración. Son muy utilizadas en proyectos Java EE, ya que permiten mejorar la estabilidad de la aplicación y la ejecución de los juegos de pruebas implementados, de forma automática.

- Herramientas de control de versiones. La mencionamos en la misma categoría que a las herramientas de compilación automáticas. No es una herramienta exclusiva de las aplicaciones Java Ee, pero si que es verdad cuando trabajamos en grandes equipo de desarrollo y con un gran numero de componentes, es necesario mantener un repositorio del código que controle las diferentes versiones del mismo.

- Herramientas de para medir la calidad. Hemos considerado estas herramientas en la sección de herramientas para pruebas y hemos indicado que básicamente miden diferentes aspectos de la calidad del código. No tienen una utilidad especial en aplicaciones Java EE, que no podamos encontrar para otras tecnologías.

- Herramientas para documentar. Ubicadas en una categoría con el mismo nombre, si que disponen de herramientas muy interesantes como Javadoc, pero del mismo modo, no aportan nada especial o diferentes, que no se pueda hacer con otras tecnologías.

- Herramientas de integración, carga y pruebas de rendimiento. Estas herramientas las hemos ubicado en herramientas para pruebas y herramientas de gestión de la configuración. En esta

ocasión si que encontramos herramientas muy específicas de la tecnología que realizan pruebas para medir la carga de nuestro servidor de nuestros servicios web, interfaces swing o interfaces web, pruebas de rendimiento, etc.

- Herramientas de seguimiento de errores. Este grupo de herramientas las hemos considerado como herramientas de gestión de la configuración y las tecnologías Java EE, al igual que cualquier otra tecnología pueden o no utilizarlas, todo y ser altamente recomendables su uso.

- Herramientas de integración continua. Hemos hablado de ellas tanto en las herramientas de pruebas, como en las herramientas de gestión de la configuración y son altamente recomendadas para su utilización con tecnologías Java EE. Ya sea por la necesidad de integrar diferentes componentes, como por los diferentes entornos con los cuales nos encontramos cuando desarrollamos con Java EE y que necesitan probar su correcto funcionamiento.

Personalmente creo que son necesarias otras herramientas para trabajar correctamente con Java EE, como son entornos integrados de desarrollo y herramientas para el modelado del dominio de la aplicación y la realización del diseño de la misma. No olvidemos que Java es un lenguaje orientado a objetos y con el que podemos desarrollar aplicaciones conceptualmente muy complejas, con tal de facilitarnos las tareas previas a su implementación necesitamos un conjunto de herramientas que eviten diseños defectuosos y errores de concepto.

Conclusiones

La clasificación que hemos utilizado para identificar las diferentes herramientas no es perfecta, como habréis observado muchas veces se solapan diferente tipos de herramientas entre ellas, pero hemos considerado que nos proporciona una visión lo suficientemente global y a su vez intuitiva, de las herramientas que podemos encontrarnos en prácticamente cualquier desarrollo del software.
No hemos querido detallar excesivamente las funcionalidades de cada grupo o categoría, del mismo modo que no hemos realizado descripciones de las herramientas ejemplo. Tampoco hemos llegado a realizar una comparativa o prueba de concepto de las herramientas mencionadas, las causas de estos vacíos son debidas al excesivo incremento de trabajo y de la extensión de este documento que habrían provocado, sin llegar a reportarnos un conocimiento mucho mayor de las mismas, y que temporalmente se podrían considerar trabajo para extensiones futuras.
Hemos finalizado este capítulo con una sección que intentaba hacer referencia al tema principal de todo el documento, especificando un poco más el uso de las herramientas en metodologías ágiles y en aplicaciones java EE.

Referencias

[1] The Institute of Electrical and Electronics Engineers, Inc. IEEE Standard Glossary of Software Engineering Terminology. February 1983

[2] Grundy, J., Hosking, J., Software Tools. Department of Computer Science, University of Auckland.

[3] Beranek, P.M. The impacts of relational and trust development training on virtual teams: an exploratory investigation. In Proceedings of the 33rd Annual Hawaii International Conference on System Science. vol.1, 2000, pp.10.

[4] Firth, R., Mosley, V., Pethia, R., Roberts, L., Wood, W., A guide to the Classification and Assesment of Software Engineering Tools. Technical Report CMU/SEI-87-TR-10 1987.

[5] Software Development Tools Directory. http://www.softdevtools.com/

[6] Tigris.org. Open Source Software Engineering Tools http://www.tigris.org/

[7] Programming Tools. http://en.wikipedia.org/wiki/Programming_software

[8] Ferguson Smart, J., Java Power Tools. O'Reilly. 2008

8. Caso práctico

En este capítulo mostramos un caso práctico de aplicación de dos metodologías ágiles en un proyecto desarrollado con las tecnologías Java EE. El principal objetivo es tener una primera toma de contacto práctica con las metodologías ágiles y sus herramientas asociadas. Las metodologías escogidas han sido Scrum Y Extreme Programming, junto con un conjunto de herramientas completamente arbitrario. A lo largo del capítulo podremos ver las diferentes fases del proyecto, documentos generados y las conclusiones del ejercicio piloto.

Contenidos de la sección

Métodos, técnicas y herramientas.

Antes de iniciar el proyecto piloto teníamos que escoger diferentes métodos, técnicas y herramientas para llevarlo a cabo, a continuación explicamos brevemente las metodologías ágiles escogidas y sus técnicas y herramientas.

Selección de las metodologías

La elección de las metodologías ágiles a utilizar en este proyecto se ha sustentado fundamental y principalmente, en la popularidad(ver selección metodologías , del capítulo 5) y la documentación, siendo Scrum y Extreme Programming las dos metodologías que cumplen estos requisitos. Se tuvieron en cuenta también otros aspectos como que ya existiesen experiencias documentadas de su uso [3].

Selección de las técnicas y métodos

Hemos seguido las indicaciones de Henrik Kniberg para la realización de proyectos con Scrum y XP, que presenta en su libro *Scrum y Xp from the trenches* [2]. De tal manera que los métodos y técnicas que utilizamos para Scrum han sido:

- La pila del producto o product backlog. Es el documento a través del cual se recogen los requisitos de los clientes.
- La pila de sprint o sprint backlog. Es el conjunto de historias, que pertenecen a la pila del producto, que vamos a realizar en el sprint.
- Estimación del esfuerzo. Cada historia de la pila del producto es estimada con lo que llamamos puntos de historia y que se corresponde a días-persona ideales.
- Gráfico Burn-down. Este gráfico ayuda a medir la productividad y detectar riesgos de mala distribución del trabajo y desviaciones temporales.
- Reuniones para cada sprint.
- Reuniones de seguimiento diarias.
- Planning Poker. Esta técnica se utiliza a la hora de realizar la estimación de las historias. Esta técnica simula el juego de Poker para obtener un estimación colectiva y cooperativa de cada historia.

Y para Extreme Programming:

- Test Driven Development:
  - Diseñar las pruebas antes que implementar las funcionalidades.
  - Todo código que pasa a producción tiene sus pruebas asociadas.

- Refactorizar. Reescribir aquellos métodos y clases, para mejorar la legibilidad del código. Esta es una técnica fundamental cuando se genera software de manera incremental y a partir de las pruebas.

- Diseño incremental. Mantener un diseño simple desde el principio e ir mejorándolo continuamente.

- Integración continua. Esta técnica nos permite compilar el código cada vez que se sube al repositorio de código, ejecutar las pruebas y desplegarlo en un entorno de producción. A su vez, puede realizar este proceso de forma automática y cada cierto tiempo, de tal manera que podemos observar los diferentes resultados de las pruebas en diferentes condiciones y horas.

Selección de las herramientas

A la hora de escoger las herramientas nos hemos ceñido a la breve experiencia de la que disponíamos y las recomendaciones de otros compañeros, seleccionando las herramientas necesarias para poder llevar a cabo las técnicas comentadas arriba. Estas son las herramientas escogidas:

1. MySQL 5.1 como servidor de base de datos.

2. Jetty 6.1.10 como contenedor Web embebido para el entorno de integración y pruebas.

3. Tomcat 5.5 como servidor de aplicaciones en los entornos de desarrollo y producción.

4. Hudson como herramienta de integración continua. Hudson se encargaba de coger la última versión del código del repositorio e invocar a Maven para que este compilase, ejecutase las pruebas y desplegase el war. Hudson monitoriza estos procesos y muestra informes de cada compilación y juego de pruebas.

5. Maven como herramienta de compilación y despliegue automático.

6. Subversion como herramienta de control de versiones y repositorio de código.

7. Inicialmente utilizamos IceScrum como herramienta de control y gestión del proyecto, pero todavía se encuentra en una fase temprana de desarrollo y tienes algunos fallos funcionales graves que hicieron que prefiriese utilizar un hoja de cálculo para gestionar la pila del producto, pilas de sprint, tareas y gráfico burn-down.

8. Eclipse Ganymede, como IDE de desarrollo, con los siguientes plugins:

    - Web Tools Platform, para el desarrollo de aplicaciones web.
    - Subclipse, para la utlización de subversión con Eclipse.
    - JUnit 4.4 como framework de desarrollo de las pruebas unitarias.
    - M2eclipse, para utilizar maven con eclipse.

Desarrollo del Proyecto

Este proyecto tiene su origen como una necesidad práctica para contrastar las prácticas, métodos y herramientas de las metodologías ágiles en un proyecto software con las tecnologías Java Enterprise Edition. Al no existir ninguna propuesta en firme respecto a la temática del proyecto, mi director en Everis propuso la realización de una aplicación web que mostrase los datos recogidos para la selección de las metodologías ágiles y se fijaron la utilización de las metodologías Scrum y XP, como metodologías de desarrollo y gestión del proyecto. Con este objetivo principal podemos ya iniciar el proyecto y especificar un conjunto de funcionalidades posibles, como pueden ser la necesidad de autenticarse, generación de gráficos, modificación del contenido, y otras nuevas funcionalidades que podamos realizar en el tiempo disponible.

Alcance

El proyecto que nos ocupa no debe alargarse temporalmente más allá del 15 de Julio, ya que debemos disponer de tiempo suficiente para evaluar la experiencia y redactar las conclusiones de la utilización de los métodos tan peculiares de Scrum. Por tanto, el proyecto tendrá una duración aproximada de un mes y medio, donde se desarrollará una aplicación web que permitirá observar los datos recogidos para la selección de las metodologías ágiles y como usuario autenticado modificarlos, añadir nuevas metodologías al estudio o eliminar metodologías ya existentes. Con estos requisitos tan básicos nos proponemos realizar un proyecto muy sencillo, que no nos ocupe excesivo tiempo y nos permita evaluar las metodologías ágiles escogidas. De la misma manera, al no establecer un conjunto de requisitos iniciales muy elevado, establecemos un clima ideal para la utilización de metodologías ágiles, pudiendo introducir variaciones y/o nuevos requisitos a lo largo del proyecto, observando de qué modo se desenvuelve la metodología.
Remarcamos que este proyecto piloto es una simulación y se engloba en un ambiente completamente ficticio, por lo que las situaciones en las que se desarrolla son excepcionales y difíciles de repetir. Una sola persona representa los mismos roles de Scrum master, product owner y equipo de desarrolladores, las reuniones diarias las realiza la misma persona y todas las tareas asociadas, los distintos equipos necesarios son virtualizados en una misma máquina. Todas estas características merman la calidad de las conclusiones que hacen referencia a la efectividad o la calidad del desarrollo del proyecto y el producto en si, pero nos permiten evaluar la aplicabilidad de las técnicas, métodos y herramientas en un proyecto Java EE y observar desde un prisma más práctico todos los conceptos que hasta momento solo hemos estudiado.

Pila de producto

La pila del producto es el documento más relevante en el desarrollo de proyectos con Scrum. Básicamente es una lista priorizada de requisitos, historias o cualquier cosas que el cliente quiere, descritas en lenguaje que el cliente pueda entender. A cada ítem de la pila lo llamamos historias.
De los diferentes campos recomendados para identificar cada historia, hemos seleccionado los siguientes:

- Item Id. Este número es un identificador auto-incremental y nos identifica cada historia de tal manera que no perdamos su pista aun cuando le cambiemos su nombre.
- Nombre de la historia. Es una descripción corta de la historia.
- Estado. Una historia puede tener tres tipos de estados, planificado, en curso y realizado. Indica en que estado se encuentra cada historia.
- Estimación inicial. Es la estimación inicial del equipo sobre la cantidad de trabajo que es necesario para implementar la historia, comparada con otras historias. La estimación la realizamos en puntos de historia, que serían equivalentes a dias-persona ideales [2]. Lo

realmente importante aquí no es que si ponemos 3 puntos de historia, esa historia se desarrolle en 3 días, sino que si otra historia esta estimada en 6 puntos, su carga de trabajo debe ser del doble que la de 3.

- Sprint. Indica en que sprint será llevado a cabo, es un número.
- Prioridad. Nos da el orden de prioridad que el propietario del producto ha establecido de acuerdo a la importancia que el cliente da a cada historia.
- Comentarios. Escribimos las indicaciones sobre la historia que nos puedan ayudar a estimarla y a dividirla en diferentes tareas.

A continuación podemos ver en la tabla 1 la pila de producto de nuestro proyecto.

Item Id	Nombre de la historia	Estado	Estimación	Sprint	Prioridad	Comentarios
1	Diseño apariencia de la aplicación.	Plan.	8	1	1	Modelo conceptual de la BBDD e insertar datos.
2	Mostrar comparativa del estudio de Metodologías.	Plan.	6	2	2	Utilización de servlets, jsp. Contenido dinámico
3	Login de usuario.	Plan.	3	3	3	
4	Añadir nuevas metodologías al estudio.	Plan.	3	3	4	Verificar campos obligatorios y formato de los datos introducidos.
5	Eliminar metodologías y todos sus datos.	Plan.	2	3	5	
6	Modificación datos introducidos.	Plan.	3	-	6	
7	Añadir papers como resultado de una búsqueda.	Plan.	6	-	7	Como se suben ficheros al servidor.
8	Consultar palabras clave de las búsquedas.	Plan.	1	-	8	
9	Top 5 metodologías ágiles.	Plan.	5	-	9	
10	Realizar automáticamente búsquedas y actualizar los datos.	Plan.	10	-	10	
11	Generar gráficos mejor documentación y mayor presencia en Internet.	Plan.	10	-	11	

tabla 1

Hemos dividido la pila de producto en tres colores diferentes que marcan las siguientes prioridades y que según Kniber [2], resulta de gran ayuda para poder establecer un precio cerrado al desarrollo de un producto con Scrum:

- Indica que son historias que deben realizarse para la primera versión del producto.
- Indica que son historias que deberían incluirse en la primera versión del producto..
- Indica que son historias optativas, que pueden hacerse mas tarde.

Sprints y pila de sprints

Scrum define cada una de las iteraciones como sprints, para los cuales se escogen un conjunto de historias de la pila del producto, creando la pila del sprint. Las historias escogidas para cada sprint serán las que se desarrollarán a lo largo de esa iteración, por tanto es importante acertar en la estimación de esfuerzo o tiempo que dedicaremos a cada historia y a la velocidad de nuestro equipo y de este modo asignar el tiempo al sprint. Los sprint suelen tener una duración bastante reducida y que no suele superar el mes o mes y medio[1], lo mas normal son tres semanas. Nosotros hemos fijado una duración de tres semanas para cada sprint, ya que el proyecto tenia una duración bastante reducida y queríamos experimentar con la ejecución de al menos tres sprints.

En las tablas 2, 3 y 4 podemos observar las pilas del primer, segundo y tercer sprint respectivamente.

Pila de Sprint 1.

Item Id	Nombre de la historia	Estado	Estimación	Sprint	Prioridad	Comentarios
1	Diseño apariencia de la aplicación.	Plan.	8	1	1	Modelo conceptual de la BBDD e insertar datos.

tabla 2

Pila de Sprint 2.

Item Id	Nombre de la historia	Estado	Estimación	Sprint	Prioridad	Comentarios
2	Mostrar comparativa del estudio de Metodologías.	Plan.	6	2	2	Utilización de servlets, jsp. Contenido dinámico

tabla 3

[1] El desarrollo iterativo según el estándar Nokia establece que las iteraciones deben tener una duración fija de menos de seis semanas [2].

Pila de Sprint 3.

Item Id	Nombre de la historia	Estado	Estimación	Sprint	Prioridad	Comentarios
3	Login de usuario.	Plan.	3	3	3	
4	Añadir nuevas metodologías al estudio.	Plan.	3	3	4	Verificar campos obligatorios y formato de los datos introducidos.
5	Eliminar metodologías y todos sus datos.	Plan.	2	3	5	

tabla 4

El procedimiento que hemos seguido para escoger que historias incluir a cada sprint utiliza el cálculo de la velocidad del equipo de desarrollo, este parámetro lo explicamos en el siguiente punto. La velocidad del equipo nos indica cuantos puntos de historia puede realizar nuestro equipo en cada sprint y por tanto, nos permite escoger que historias podemos desarrollar en cada sprint. La selección de las historias sigue el orden establecido por el propietario del producto a través la prioridad, es decir que siguiendo la prioridad establecida, para cada sprint escogemos tantas historias como nuestra velocidad nos permita.

Estimación del esfuerzo y cálculo de la velocidad

Uno de los puntos claves en el desarrollo de proyectos con Scrum es la estimación del esfuerzo y el cálculo de la velocidad del equipo de desarrollo. En el proyecto piloto que nos ocupa hemos utilizado la técnica de planning poker para estimar el esfuerzo necesario para cada una de las historias de la pila del producto. Tal y como hemos comentado al principio de este capítulo, las unidades de esfuerzo se llaman puntos de historia y se corresponden con días-persona ideales. Planning poker es una técnica de estimación de esfuerzo, la cual se realiza con trece cartas o fichas que contienen los siguientes símbolos 0, 1/2, 1, 2, 3, 5, 8, 13, 20, 40, 100, ? y una "taza de café" . Todos los miembros del equipo de desarrollo escogen una carta para cada historia y la ponen boca abajo sobre la mesa, cuando todos han escogido carta, se les da la vuelta y se observan las diferentes estimaciones. Se discuten las divergencias mostradas y se repite este proceso hasta llegar a un acuerdo sobre el esfuerzo de la historia. En el proyecto que nos ocupa utilizamos la herramienta IceScrum, que permite utilizar esta técnica a través de Internet. Esta técnica es muy interesante ya que permite una estimación colaborativa y no condicionada por el desarrollador que mayor conocimiento tiene de la historia.

Con las historias están estimadas y priorizadas (la prioridad la establece el propietario del producto), seleccionamos, en cada reunión de inicio de sprint, las historias en función del cálculo de la velocidad. El cálculo de la velocidad de nuestro equipo de desarrollo se puede hacer de diferentes maneras, normalmente basta con utilizar la velocidad del equipo en el anterior sprint, pero para el primer sprint tenemos que estimarlo de alguna manera y lo hemos hecho mediante el cálculo de recursos y el factor de dedicación [2] . En el anexo 4 hemos ubicado las minutas de las reuniones simuladas y podemos encontrar el cálculo de la velocidad para el proyecto piloto en cada sprint. El cálculo de recursos es tan sencillo como considerar cuantas personas forman nuestro equipo de desarrollo, cuantos días durará el sprint y cuantos días estarán disponibles cada una de las personas de nuestro equipo de desarrollo. En el proyecto piloto solo disponíamos de una persona para el desarrollo del proyecto y la duración de los sprints estaba fijada en 15 días, considerando que no faltará ningún día de los quince del sprint, el cálculo de recursos es 15 días * 1 persona = 15 días-persona disponibles. Este resultado no tiene en

cuenta el factor de dedicación, es decir que nuestro equipo puede que este también en otros proyectos y por tanto su dedicación no será total, o que considere que las estimaciones iniciales son muy optimistas o que van a encontrar muchos impedimentos en el proyecto. Normalmente, el factor de dedicación se obtiene del último sprint realizado, pero si es el primero que realizamos, podemos considerar una estimación de los diferentes miembros del equipo (parecido a como hemos estimado las historias), en nuestro proyecto hemos considerado un factor de dedicación del 50% y por tanto la velocidad de nuestro equipo es 15 *0,5 = 7'5 puntos de historia. Hemos escogido un factor de dedicación del 50% porque es exactamente la dedicación que hemos destinado al proyecto piloto, media jornada y ¡creemos que las estimaciones que hemos hecho son muy buenas!

Diagrama Burn-Down

Los diagramas burn-down nos permiten observar los progresos de nuestro proyecto de una forma muy intuitiva y corregir las diferentes señales de alarma que nos indica, es una representación gráfica de avance del sprint. A continuación podemos ver en las figuras 1, 2 y 3 los diagramas burn-down de cada uno de los sprint, y una breve explicación de como ha evolucionado ese sprint según la lectura del diagrama. Es un diagrama realizado con MS Excel y muestra en el eje de ordenadas (Y) el trabajo restante en puntos de historia, inicialmente tenemos el total de los puntos de historia del sprint y si todo ha ido correctamente al final no deberíamos tener ninguno. En el eje de abscisas (X) representamos la evolución temporal, cada barra es un día, hasta los quince días que dura el sprint.

Diagrama Burn-Down del Sprint 1.

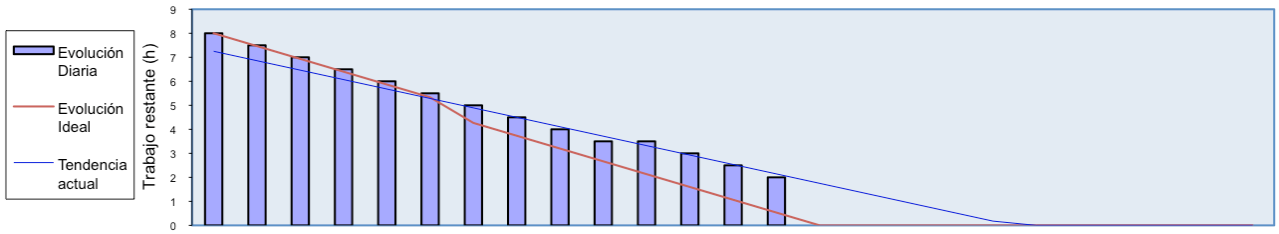


fig1

En este diagrama podemos observar que no hemos seguido la evolución ideal y no se han realizado todas las tareas planificadas. Este peligro lo advertimos el día siete del sprint y decidimos dividir una de las tareas de tal manera que se pudiese acabar en el siguiente sprint. La tarea que dividimos fue preparación del entorno, realizando nada más la instalación y configuración de las máquinas virtuales y dejando para el siguiente sprint las tareas de instalación y configuración de las herramientas. La velocidad del equipo en este sprint fue de 6 puntos de historia.

Diagrama Burn-Down del Sprint 2.

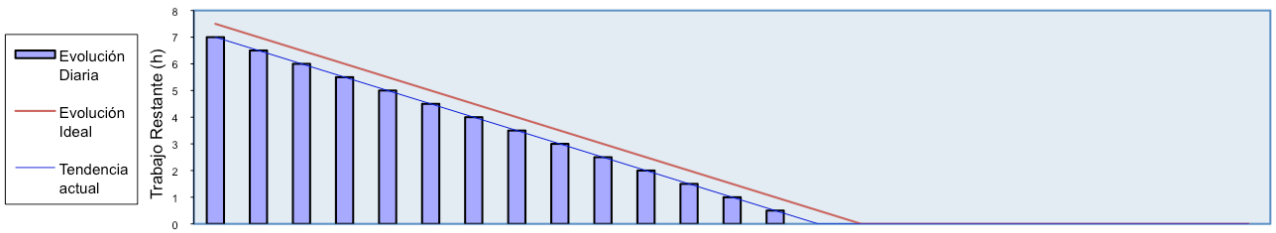


fig 2

En este caso podemos observar un diagrama de burn-down perfecto, en el cual se han cumplido todas las tareas planificadas y se ha seguido la evolución ideal. En el sprint 2 hemos cumplido con todas las tareas planificadas y lo hemos hecho con una velocidad de 7'5 puntos de historia.

Diagrama Burn-Down del Sprint 3.

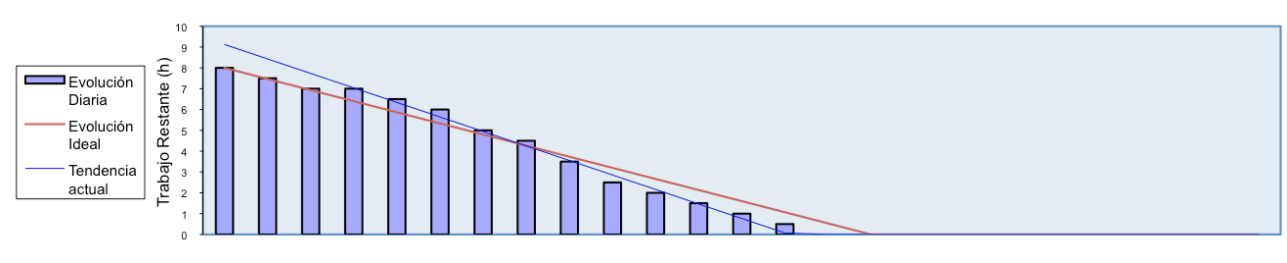


fig 3

En este diagrama podemos observar como al cuarto día del sprint, el ritmo de trabajo era insuficiente y de haber seguido así no habríamos concluido todas las tareas, pero la velocidad se corrigió y todas las tareas finalizaron en el tiempo requerido. La velocidad del equipo al final de este sprint ha sido de 8 puntos de historia.

Tareas

A la hora de organizar, repartir y trabajar sobre las historias de la pila del sprint, se suele realizar una división de cada historia en diferentes tareas. Es importante observar la diferencia entre tarea e historia, mientras que una historia es un entregable y es responsabilidad del dueño del producto, una tarea no es entregable y la responsabilidad es del equipo de desarrollo. Una historia suele dividirse en diferentes tareas. Veamos las diferentes tareas especificadas para cada historia y en que sprint se han realizado (tabla 5).

Sprint	Nombre de la tarea	Historia la que pertenece	Estimación
1	Modelo conceptual de la base de datos.	1	1
1	Introducción datos en la base de datos.	1	2
1	Diseño GUI y disposición de los datos.	1	2
1	Preparación del entorno - Instalación máquinas virtuales, subversion, servidor Integración continua, ...	1	3
2,3	Definir y escribir pruebas.	2,3,4,5	2
2	Formación JSP y Servlets.	2	2
2	Diseño de la arquitectura de la aplicación.	2	1
2	Obtener datos comparativa de la base de datos.	2	1,5

Sprint	Nombre de la tarea	Historia la que pertenece	Estimación
3	Modificar diseño de la página al añadir login.	3	1
3	Diseñar botones y comportamiento.	3,4,5	2
3	Implementar llamadas a base de datos.	3,4,5	4

tabla 5

Como podéis observar, una de las tareas recurrentes es la creación de las pruebas, esto es debido a que estamos practicando Test Driven Development y es una condición fundamental para su desarrollo. La tarea de preparación del entorno fue dividida en dos, para realizarse en las dos primeras iteraciones.
Algunas de las tareas especificadas generan artefactos que pueden ser interesante o relevantes par el lector (modelo conceptual BBDD o arquitectura del sistema), estos artefactos los pueden encontrar en el anexo 4.

Reuniones y Roles

Scrum es una metodología ágil que prescribe el uso de reuniones como medida de control del progreso, productividad, estimación del esfuerzo, planificación, etc. Si queremos practicar Scrum tal y como la metodología nos recomienda, debemos realizar una reunión antes y después de cada sprint y una reunión de seguimiento cada día. A la hora de realizar este proyecto piloto, una de las tareas más difíciles de simular han sido sin duda la realización de las reuniones, nuestra manera de proceder ha sido elaborar un acta o minuta de las principales reuniones, las reuniones de inicio de sprint y fin de sprint, y hemos realizado las tareas que deberían hacerse en las reuniones diarias (actualizar diagrama burn-down), pero sin ningún tipo de acta. Las reuniones diarias deben ser cortas, concisas y simplemente controlar que todo va según lo planificado, si detectamos algún tipo de problema, lo hemos reflejado en el acta de final de sprint.
En el anexo 4 hemos incluido el acta de estas reuniones, así como otro tipo de artefactos generados en el proyecto.
Otro punto muy importante para el desarrollo de Scrum y que se hace latente a la hora de simular reuniones, es la distribución de los roles o responsabilidades en Scrum. Evidentemente esto no ha sido posible, ya que una sola persona ha tenido que realizar todas las tareas y roles. Es una carencia que ha tenido este proyecto y que no hemos podido solventar, provocando claros cruces de intereses, ya que la misma parte era propietaria del producto y equipo de desarrolladores, al menos las reuniones no eran largas y se llegaba rápidamente a un punto que agradaba a todas las partes.

Extreme programming

Al inicio de este capítulo hemos especificado que técnicas escogíamos de cada una de las metodologías seleccionadas. Para extreme programming hemos seleccionado Test Driven Development, diseño incremental, refactorización e integración continua.

1. *Test Driven Development.*

El principio por el cual se rige esta técnica/metodología es el desarrollo dirigido a través de las pruebas y de este modo hemos intentado realizar el desarrollo de este prototipo.
En la tabla 6 detallamos el conjunto de pruebas que definimos para cada historia y a partir de las cuales desarrollamos todo el producto. Hay muchas más pruebas que no hemos creado, por ejemplo de carga y tiempo de respuesta, ya que no tenemos ningún requisitos no funcional al respecto. Las pruebas realizadas han sido creadas mediante el framework Junit, con el que no habíamos trabajado

anteriormente, este hecho a provocado que fuésemos aprendiendo a medida que hacíamos las pruebas, que y como podíamos hacer las pruebas. Este proyecto ha sido demasiado sencillo como para probar la potencia de esta técnica, pero nos ha servido para ver como aplicarla, obligarnos ha hacer un importante cambio de mentalidad a la hora de desarrollar y observar como obteníamos una aplicación un poco diferente a lo que estábamos habituados.

Historia	Pruebas escritas
1. Diseño apariencia aplicación	No hay pruebas definidas.
2. Mostrar comparativa del estudio de Metodologías.	• Prueba conexión con la base de datos. • Prueba creación de una nueva metodología vacía. • Prueba creación de una metodología con datos. • Prueba para obtener todas las metodologías de la base de datos. • Prueba para obtener metodologías inexistentes.
3. Login usuario.	• Prueba creación usuario existente. • Prueba creación usuario inexistente. • Prueba borrar usuario existente. • Prueba borrar usuario inexistente.
4. Añadir nuevas metodologías al estudio.	• Prueba insertar en base de datos metodología con datos correctos. • Prueba insertar en base de datos metodología con nombre repetido. • Prueba insertar en base de datos metodología con clave internet repetida y mismo buscador. • Prueba insertar en base de datos metodología con clave de paper repetida y mismo buscador. • Prueba insertar en base de datos metodología con clave de paper repetida y diferente buscador. • Prueba insertar en base de datos metodología con buscador existente. • Prueba insertar en base de datos dos metodologías correctas consecutivas.
5. Eliminar metodologías y todos sus datos.	• Prueba borrar metodología existente. • Prueba borrar metodología inexistente.

tabla 6

2. Diseño incremental.

Hablamos de diseño incremental cuando partimos de un diseño inicial sencillo y vamos poco a poco incrementado su complejidad y tamaño.

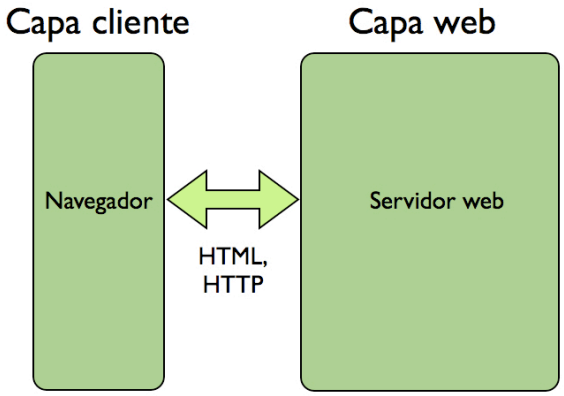


fig 4

El diseño de esta aplicación es muy sencilla y nos hemos restringido a generar un diseño de la arquitectura del mismo. Inicialmente tan solo teníamos una página web estática y por tanto el diseño estaba compuesto por un navegador y un servidor web (ver figura 4). En el segundo sprint añadíamos dinamismo, obtenemos los datos de una base de datos, necesitamos una capa de datos y sustituimos el servidor web por un servidor web (ver figura 5) para poder trabajar con los Jsp.

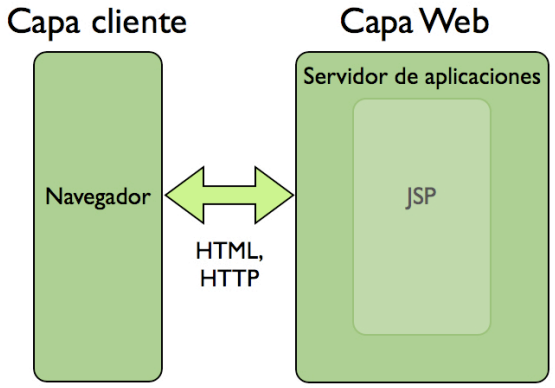


fig 5

En el tercer y último sprint añadimos la lógica y controlamos las peticiones con servlets, ver figura 6.

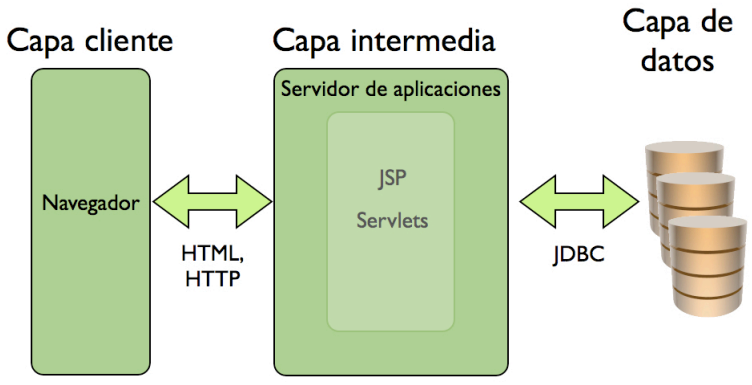


fig 6

3. *Refactorizar.*

Gracias a la utilización de un muy buen entorno de desarrollo integrado (Eclipse), refactorizar es una tarea amena y que prácticamente no me ha requerido mucho tiempo. Eclipse permite que cada vez que modificas una función, clase o atributo, automáticamente busca todas las referencias en tu proyecto y con un asistente te las muestras y pregunta si deseas modificarlas. No tuve que utilizarlo muchas veces, pero estás muy tranquilo disponiendo de una herramienta de este calibre.

4. *Integración continua.*

Esta ha sido una de las técnicas que más trabajo nos ha ocasionado, ya que implicaba configurar todo un entorno de desarrollo formado por dos máquinas virtualizadas y sus herramientas asociadas.
En la figura 7 podemos observar la organización de las diferentes máquinas configuradas. Se ha utilizado un portátil con Mac OS X como equipo de desarrollo, al cual se le han instalado dos máquinas virtuales con Ubuntu 8.0.4 que hacen las veces de entorno de pruebas y entorno de producción. De este modo hemos podido probar las herramientas en una situación más o menos real.

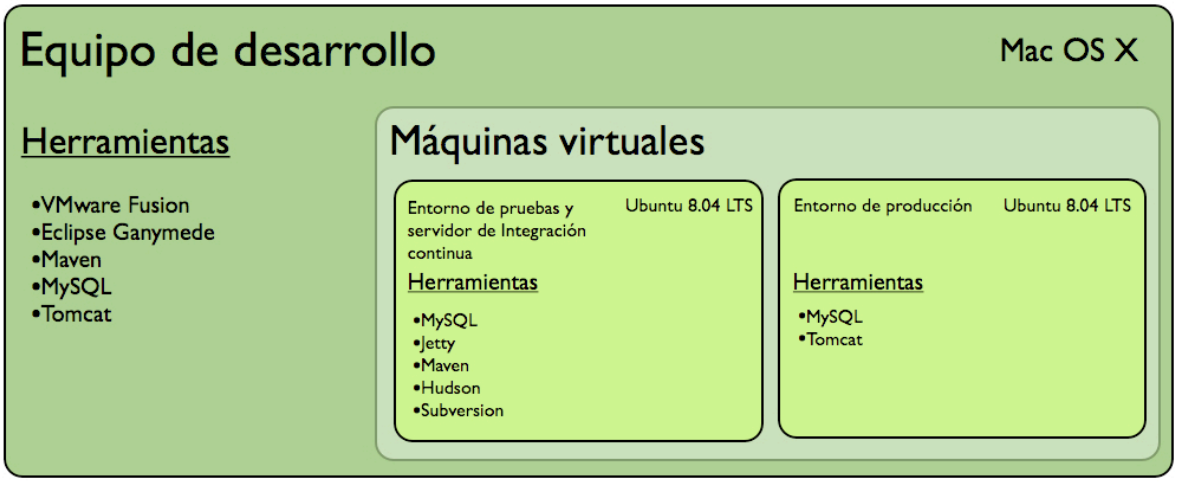


fig 7

La configuración con la que hemos trabajado nos ha permitido desarrollar en el equipo de desarrollo, actualizar el repositorio de código en el entorno de pruebas y ejecutar las pruebas. El servidor de integración continua ejecuta las pruebas y si son correctas, despliega la aplicación en el entorno de producción, en caso contrario la herramienta de integración continua (Hudson) guarda una estadística de las veces que se ha compilado y ejecutado las pruebas con éxito, que errores han aparecido, cuando fue la última vez que se compiló y ejecutaron las pruebas, etc. Es de este modo que hemos implementado y utilizado la técnica de integración continua.

Definición de fin del proyecto

Consideramos el proyecto piloto acabado cuando se han realizado los tres sprints. Como propietario del producto puedo decidir cuando hemos alcanzado este estado y para la finalidad de este proyecto piloto, creemos que el trabajo realizado es más que suficiente, por lo tanto con las historias seleccionadas y realizadas podemos dar por concluido el proyecto. El concepto de proyecto terminado es muy importante en cualquier proyecto, pero para Scrum es especialmente relevante que tanto el propietario del producto como el equipo de desarrollo tengan claro que estado define el proyecto como finalizado. La mejor manera de conseguir este objetivo es definir para cada historia el estado terminado

e ir comprobando si todas las historias requeridas por el propietario del producto han alcanzado este estado, si es así el proyecto ha finalizado.

Conclusiones

Este proyecto piloto no nos permite realizar una valoración que se ajuste a la realidad de las metodologías escogidas y no era ese su objetivo, sino realizar un caso práctico de aplicación de las metodologías ágiles y sus herramientas. Con tal de poder obtener unos resultados empíricamente válidos, deberíamos haber realizado un proyecto en un entorno real, con un proyecto real y un equipo completo. A pesar de estas limitaciones, si que podemos expresar nuestra valoración personal de la experiencia y destacar qué consideramos aplicable a proyectos reales y qué deberíamos tener en cuenta.
La valoración de la experiencia es muy positiva, consideramos Scrum una metodología que obliga a las personas que la utilizan a tomar decisiones consensuadas, cada día se controla que no hayan desviaciones de la planificación original y un propietario del producto cuida del producto. Todo parece bastante acertado, sino fuese porque podemos considerar excesivo el número de reuniones que promueve, un grado de visibilidad que puede resultar incomodo para los desarrolladores y unas restricciones que no gustan a todos los clientes (un sprint es inamovible por parte del cliente, el equipo puede decidir incluir o extraer historias o tareas, tal y como hemos visto en el proyecto). Scrum no lo aplicaría a proyectos con un alto grado de confidencialidad, pero parece encajar perfectamente con Extreme Programing o al menos con algunas de sus técnicas. A la hora de escoger la metodología de desarrollo para el proyecto piloto, buscamos información practica y sobretodo guías para novatos, que nos ayudasen con los típicos problemas iniciales, fue en este proceso que encontramos el libro de Kniberg [2], donde promulga el uso de dos metodologías ágiles y nos pareció genial, ya que nos permitía practicar con más técnicas y métodos que si tan solo hubiésemos escogido una.
Extreme Programming es una metodología que funciona muy bien tanto si se utiliza al completo o si se trocea y escogen las técnicas que mejor se adecuen a nuestra situación De este modo hemos podido utilizar Scrum como la metodología de gestión ágil de proyecto que es y XP como la metodología de desarrollo ágil que conocemos. De todas las técnicas que hemos utilizado de XP , Test Driven Development ha sido la que más gratamente nos ha sorprendido. TDD sin poder llegar a aplicarla tal y como se merece, nos ha permitido experimentar con un gran número de herramientas, conocer como funciona un servidor de integración continua, aprender a utilizar frameworks de desarrollo de pruebas y sobretodo y quizás lo más importante, ha hecho que veamos el desarrollo del software desde otro prisma. Siempre que desarrollamos una aplicación, las pruebas se encuentran en la última fase, tienen que probar que lo que has hecho funciona y no tiene errores, pero que pasa si piensas al revés, hacer unas pruebas que validen los requisitos de tu aplicación (no es más que implementar los criterios de aceptación de tus clientes) y luego generar el código que sea capaz de pasar esas pruebas. No es sencillo pensar de esta manera y he de confesar que en el proyecto no he sido capaz de hacerlo siempre, alguna clase se te escapa y estas tan acostumbrado a hacerlo al revés ...

Scrum y XP son dos metodologías compatibles, no lo he demostrado con este proyecto piloto, pero pueden encontrar un gran número de ejemplos [2][3] en los que han sido utilizados satisfactoriamente. A lo largo de este sub-proyecto hemos aprendido como utilizar sus técnicas y métodos y que en un proyecto muy sencillo, funcionan, pero esto es todo lo que podemos decir de este ejemplo sin caer en especulaciones.

Referencias

[1] Juan Palacio, Flexibilidad con Scrum, principios de diseño e implantación en campos Scrum. SafeCreative. Edición Octubre 2007

[2] Kniberg, H., XP and Scrum from the Trenches. How we do Scrum. InfoQ 2007.

[3] Jensen. Cross-continent development using Scrum and XP. Extreme Programming and Agile Processes in Software Engineering. 4th International Conference, XP 2003. Proceedings (Lecture Notes in Computer Science Vol.2675), 2003, p 146-53

9. Planificación y estudio económico

Este capítulo muestra una repartición de las tareas desarrolladas a lo largo del proyecto, en un periodo temporal de unos seis meses de duración. También se realiza una estimación de los costes que tendría el trabajo desarrollado a lo largo de este documento, en el mundo laboral. La estimación de costes resulta especialmente interesante ya que se ha realizado tareas de investigación y desarrollo, por lo que mostramos estas dos partes bien diferenciadas.

Contenidos de la sección

Planificación

La primera tarea que realizamos en este proyecto fue la determinación de los objetivos, para acto seguido dividirlos en tareas y estimar el tiempo necesario para realizarlas. Este método de planificación no nos ha dado muy buenos resultados, como ya habréis podido leer en la introducción, se nos planteaba un proyecto muy ambicioso y estimamos un tiempo insuficiente para la realización de todos los objetivos. A continuación mostramos la planificación inicial que indicaba como fecha de finalización del proyecto el 15 de Julio, ver figura 1.

Id	Nombre de tarea	Duración	Comienzo
febrero 2008			
3	Realizar clasificación de los proyectos tipo que utilizan Java EE	12 días	lun 11/02/08
4	Especificar las características fundamentales de cada proyecto y caracterizar	14 días	mié 27/02/08
marzo 2008			
4	Especificar las características fundamentales de cada proyecto y caracterizar	14 días	mié 27/02/08
16	Evolución histórica, tipos de metodologías	3 días	mar 18/03/08
17	Necesidad de las metodología en proyectos software	5 días	vie 21/03/08
19	Búsqueda de información de metodologías ágiles	5 días	lun 31/03/08
abril 2008			
19	Búsqueda de información de metodologías ágiles	5 días	lun 31/03/08
20	Listado de metodologías	1 día	lun 07/04/08
21	Selección de metodologías	23 días	mar 08/04/08
mayo 2008			
21	Selección de metodologías	23 días	mar 08/04/08
22	Caracterizar metodologías seleccionadas	2 sem.	vie 09/05/08
45	Selección de metodología de desarrollo parta proyecto piloto	1 día	vie 09/05/08
46	Adquirir conocimietos sobre la metodología	1 sem	lun 12/05/08
47	Realizar el proyecto	1,5 mes	lun 19/05/08
7	Buscar herramientas	2 días	mié 21/05/08
8	Análisis de las herramientas	3 días	vie 23/05/08
24	Búsqueda de información metodología COM	2 días	vie 23/05/08
25	Descripción de las características	3 días	mar 27/05/08
9	Clasificación de las herramientas	3 días	mié 28/05/08
36	Determinar los principles factores de riesgo desarrollo del software	4 días	jue 29/05/08
27	Búsqueda de información metodologías tradicionales	2 días	vie 30/05/08
junio 2008			
47	Realizar el proyecto	1,5 mes	lun 19/05/08
36	Determinar los principles factores de riesgo desarrollo del software	4 días	jue 29/05/08
27	Búsqueda de información metodologías tradicionales	2 días	vie 30/05/08
28	Caracterización metodología tradicional	2 días	mar 03/06/08
37	Determinar puntos clave para realizar proyectos Java EE con éxito	3 días	mié 04/06/08
38	Extraer puntos más importante normativas ISO 9000	3 días	lun 09/06/08
39	Determinar que parámetros, métricas y métodologías sirven para determinar la calidad del Software	5 días	jue 12/06/08
40	Determinar parámetros software Java EE	2 días	jue 19/06/08
41	Determinar parámetros metodologías ágiles	2 días	lun 23/06/08
42	Determinar parámetros metodología COM	2 días	mié 25/06/08
43	Determinar parámetros metodologías tradicionales	2 días	vie 27/06/08
48	Fin del proyecto piloto	0 días	vie 27/06/08
julio 2008			
30	Comparativa de metodologías	2 días	mar 01/07/08
11	Comparativa de herramientas	2 días	jue 03/07/08
31	Asociar metodologías con proyectos Java EE	2 días	jue 03/07/08
12	Asociar conjuntos de herramientas, proyectos y metodologías.	3 días	lun 07/07/08
33	Especificar defectos metodologías actuales	3 días	lun 07/07/08
34	Proponer soluciones	7 días	lun 07/07/08
13	Prueba de producto herramientas	3 días	jue 10/07/08
49	Fin del proyecto y memoria	0 días	lun 14/07/08

fig 1

La tarea Realizar el proyecto, hace referencia a el proyecto Piloto que hemos desarrollado como caso práctico de estudio. Este listado de tareas ha sido generado a través de la herramienta MS Project, pero no mostraremos como es típico, el diagrama de Gantt, ya que no aporta ninguna información nueva que no se este mostrando en esta lista de tareas y en este caso concreto, es difícil de interpretar debido al gran número de tareas que existen.

A mediados del mes de Abril ya vimos que no iban a poderse realizar todas las tareas y decidimos enfocar el proyecto a la comparativa de metodologías y el proyecto piloto. Corregimos los objetivos del proyecto y las tareas asociadas, manteniendo los objetivos principales y eliminando cualquier tarea que no fuese fundamental y obtuvimos la distribución final de las tareas del proyecto, que es la que mostramos en la figura 2.

Id	Nombre de tarea	Duración
febrero 2008		
3	Realizar clasificación de los proyectos tipo que utilizan Java EE	12 días
4	Especificar las características fundamentales de cada proyecto y caracterizar	14 días
31	Elaborar memoria	128 días
marzo 2008		
4	Especificar las características fundamentales de cada proyecto y caracterizar	14 días
31	Elaborar memoria	128 días
7	Evolución histórica, tipos de metodologías	4 sem.
abril 2008		
31	Elaborar memoria	128 días
7	Evolución histórica, tipos de metodologías	4 sem.
8	Necesidad de las metodología en proyectos software	5 días
10	Búsqueda de información	5 días
11	Listado de metodologías	1 día
mayo 2008		
31	Elaborar memoria	128 días
12	Selección de metodologías	23 días
28	Selección de metodología (SCRUM + XP)	3 días
29	Desarrollo del proyecto	47 días
junio 2008		
31	Elaborar memoria	128 días
12	Selección de metodologías	23 días
29	Desarrollo del proyecto	47 días
13	Caracterizar metodologías seleccionadas	2 sem.
15	Búsqueda de información	2 días
16	Caracterización metodología tradicional	2 días
18	Determinar los principles factores de riesgo desarrollo del software	4 días
20	Determinar que parámetros, métricas y métodologías sirven para determinar la calidad del Software	5 días
19	Determinar puntos clave para realizar proyectos Java EE con éxito	4 días
21	Determinar parámetros software Java EE	4 días
julio 2008		
31	Elaborar memoria	128 días
29	Desarrollo del proyecto	47 días
21	Determinar parámetros software Java EE	4 días
22	Determinar parámetros metodologías ágiles	1 sem
24	Realizar comparativa de metodologías	2 sem.
26	Clasificación e identificación herramientas desarrollo software	1 sem
agosto 2008		
31	Elaborar memoria	128 días

fig 2

La tarea Desarrollo del proyecto hace referencia al proyecto o caso de estudio práctico. Como podemos observar los tres primeros meses siguen la planificación inicial, pero a partir de aquí ya empezamos a asumir que muchas tareas no podrían realizarse y el proyecto cambió de enfoque.

Finalmente hemos concluido los objetivos principales del proyecto, destacando que la planificación inicial fue muy ambiciosa y que no estimaba bien la magnitud de las tareas planteadas y esta fue la principal causa de la desviación de los objetivos iniciales marcados.

Estimación de costes

Los costes asociados a este proyecto se pueden calcular de diferentes maneras, una de ellas es tan simple como expresar el coste real del mismo a través del convenio de colaboración con la empresa. Otra manera es simular que las tareas desarrolladas se hubiesen cobrado a precio de mercado y por tanto, los costes varían considerablemente. Dentro de esta última posibilidad podemos observar que el proyecto esta compuesto por dos partes, las cuales son diferentes a la hora de estimar los costes. La primera parte equivale a tareas de investigación o de consultoría, mientras que la segunda parte es un mini proyecto al cual se le pueden aplicar roles clásicos, con los costes típicos. Veamos cada una de las propuestas de estimación que costes nos muestran.

1. *Coste real estimado.*

Este es el coste que ha tenido que pagar la empresa por la realización de este proyecto y que es el resultado de un convenio de colaboración empresa-universidad. La fórmula y el cálculo son muy sencillos, el contrato estipula un total de 960 horas a un precio de 7,5 € por hora, esto nos da un total de 7.200 €. La empresa debe pagar otros conceptos a la universidad que ascienden al 14 ’ 7% del total limpio atribuido al proyectista, por tanto 1.058,4 € mas. La cantidad total real que este he proyecto ha supuesto para la empresa es superior a los 8.258,4 € calculados, ya que ha puesto a disposición del

proyecto un portátil de renting y material de oficina, no contamos gastos como la luz, agua, café o la conexión a internet. Pero si consideramos el servicio de alquiler del portátil, sin depreciación del valor del mismo, de 90 € mensuales[1], 90 * 6 = 540 €.
Por tanto el precio real estimado esta alrededor de los 8.798 €

2. *Coste simulado.*

Para hacer este cálculo dividiremos el proyecto en las dos partes que hemos indicado, consultor y "mini-proyecto."

Si consideramos el precio por hora de un consultor[2] es de 25,43 €/hora (este cálculo lo explicamos detalladamente en el cálculo de los costes de los roles del proyecto piloto)y el trabajo que ha realizado el consultor son todas las tareas no relacionadas con el proyecto piloto, hemos de restarle a las 960 horas del desarrollo del proyecto las horas del proyecto piloto. El proyecto piloto ha ocupado un total de 45 días de desarrollo y un día extra por cada reunión, que suman 49 días totales. De los cuales 30 días de desarrollo han sido a jornada partida (4 horas por día) y los cuatro día de reuniones también han sido jornada partida, el resto son jornadas con un 75% de dedicación (6 horas al día). Finalmente, si sumamos todas las horas obtenemos un total de 226 horas[3] dedicadas al proyecto piloto. Es decir, que el trabajo del consultor ha sido desarrollado durante 960 h - 226 h = 734 h. El coste asociado al consultor es de 734 horas * 25,43 €/H = 18665 €.

Ahora procedemos al cálculo de costes del proyecto piloto, a partir de los roles que han participado en el mismo. Los diferentes roles que identificamos en el proyecto son:

- Scrum Master.
- Propietario del producto.
- Equipo de desarrollo. Según los roles de extreme programming.
  - Programador.
  - Probador.

Ahora partiendo de el número de horas dedicadas al proyecto (226), hacemos una estimación de los costes, partiendo de salarios aproximados obtenidos de la herramienta Infojobs Trends[4]. Recordamos que es una estimación por tanto los resultados deben considerarse como tales. La media española de horas trabajadas semanalmente es de 40, si el año tiene 52 semanas y trabajamos 48 semanas al año (1 mes de vacaciones, no contamos fiestas, ni días personales, etc.) obtenemos 48*40 = 1920 horas anuales trabajadas. Este valor nos sirve para obtener el coste por hora aproximado de cada rol.

Scrum Master es equivalente a un gestor de proyectos, el salario medio es de 32000 € si calculamos un 32% de incremento de coste para la empresa, son 42240 € anuales. 42240 € / 1920 = 22 €/hora.

---

1 Información estimada a partir de los datos de la página web http://www.rentacomputer.com/rentals/laptop.asp, precio de alquiler de un portátil en Las Vegas 140 $, precio del Dollar 0,6€ y modelo del portátil es un HP Intel Single Core con XP.

2 http://salarios.infojobs.net/resultados.cfm?sueldo=consultor&o_id=2

3 34 días al 50%, son 34 días * 4 horas por día = 136 y 15 días al 75%, son 15 días * 6 horas por día = 90. 136 h + 90 h = 226 horas totales del proyecto.

4 Propietario del producto http://salarios.infojobs.net/resultados.cfm?sueldo=product+manager
Scrum master http://salarios.infojobs.net/resultados.cfm?sueldo=proyectos&o_id=2
Programador j2ee http://salarios.infojobs.net/resultados.cfm?sueldo=programador+j2ee&o_id=2
Probador j2ee http://salarios.infojobs.net/resultados.cfm?sueldo=sistemas&o_id=
Consideramos coste de empresa un 32 % mas del salario indicado.

Propietario del producto es equivalente a un product manager que tiene un salario promedio similar al de un gestor de proyectos y por tanto su coste horas estimado para la empresa es de 22 €/hora.
Un programador j2ee tiene un salario medio de 28.000 €, repitiendo la misma operación que hemos realizado con el salario de un Scrum Master obtenemos 14,6 €/h.
Un probador es el encargado de configurar y mantener las herramientas, ademas de ejecutar las pruebas funcionales, si observamos su salario medio (26.000) y realizamos el cálculo por horas, obtenemos un coste de 13,5 €/h.

A continuación mostramos en la tabla 1, la estimación de costes teniendo en cuenta la dedicación de cada rol en el proyecto y el coste estimado. Y finalmente en la tabla 2 podemos ver el coste total de la realización del proyecto piloto.

Rol	Dedicación	Coste/ horas	Coste total
Scrum Master	20%(45,2 h)	22 €/h	994,40 €
Propietario del producto	30%(67,8 h)	22 €/h	1.491,60 €
Programador	80%(180 h)	19,25 €/h	3.465 €
Probador	30%(67,8 h)	17,88 €/h	1.212,64 €

tabla 1

Coste total estimado del proyecto piloto
7.163 €

tabla 2

Por tanto, considerando el coste que le habría presentado realizar este proyecto a la empresa internamente, sin realizar un convenio de colaboración con la empresa habría sido (ver tabla 3):

Coste total del proyecto
25.828 €

tabla 3

Si el coste real estimado ha sido de 8798 €, podemos asegurar que hemos ahorrado a la empresa donde se ha realizado el proyecto un total de 17030 €.

Conclusiones

En este capítulo hemos presentado un estudio económico de las tareas desarrolladas a lo largo de todo el proyecto, y hemos presentado las diferentes planificaciones que hemos realizado para el mismo. La conclusión que podemos extraer de la primera e infructuosa planificación es que nos plantemos objetivos excesivamente ambiciosos, al menos para el tiempo del cual disponíamos. De todo modos estamos contentos con los resultados obtenidos y de haber sabido reaccionar a tiempo y replanificar las tareas de acuerdo al tiempo disponible. Ha resultado interesante buscar información sobre los salarios que perciben roles tan específicos y poco extendidos como el de Scrum Master, al menos en España no se pueden encontrar un número relevante de ofertas a este tipo de roles y a pesar de contactar con un experimentado Scrum Master, hemos llegado a la conclusión que lo más recomendable era equipararlo a la posición del mercado español más cercana, el gestor de proyectos.
No soy partidario de la realización de estas estimaciones de costes, ya que los resultados que obtenemos son muy lejanos a la realidad, al menos en este estudio hemos podido presentar unos costes reales del proyecto, que se aproximan ciertamente a los costes que la empresa ha tenido que sufragar para obtener este documento. También es interesante indicar que los costes que hemos estimado, siempre han sido a nivel interno y ese es el motivo que trabajemos con salarios brutos anuales y porcentaje de incremento sobre el coste para la empresa. Si el proyecto se hubiese realizado para otra empresa, como un trabajo externo, los costes habrían variado sustancialmente, todos sabemos que el salario que percibe un consultor en su empresa no se suele parecer al que percibe la empresa por sus servicios en otra empresa o el que se suele presupuestar.

10. Conclusiones

Este es un capítulo diferente al resto de los que presentamos en el documento. No tiene una sección de conclusiones, no de referencias bibliográficas y su objetivo es presentar la valoración más personal de la realización del proyecto, los objetivos cubiertos y conocimientos aplicados y adquiridos..

Contenidos de la sección

Objetivos cubiertos

Los objetivos principales del proyecto han sido satisfechos, pero los objetivos que inicialmente establecimos tuvieron que ser replanteados para que el proyecto pudiese finalizarse. Si nos centramos en los objetivos que hemos enumerado en la introducción, concretamente los que determinamos que se desarrollan en el documento, podemos determinar las tareas que se han llevado a cabo para darlos como satisfechos:

1. *Entender y documentar las características fundamentales de los proyectos software realizados con JEE.*

Hemos introducido los conceptos de proyectos y proyectos software, enumerándolas características fundamentales de la plataforma Java EE, identificado los diferentes escenarios y anatomías que nos podemos encontrar en las aplicaciones desarrolladas con Java EE, detectado los roles típicos de los proyectos Java EE y realizado un estudio de las diferentes aplicaciones empresariales existentes en el mercado, a partir de las soluciones ERP de diferentes empresas del mercado.
Con este conjunto de tareas consideramos el objetivo satisfecho.

2. *Seleccionar un número de metodologías ágiles para su estudio.*

Hemos investigado el origen de las metodologías, el concepto de ágil y tradicional, presentado las características de las metodologías tradicionales, identificado un conjunto de criterios para la selección de metodologías ágiles, realizado un estudio de las diferentes metodologías ágiles existentes en función de los criterios encontrados y la selección de seis metodologías ágiles para su estudio. Finalmente hemos caracterizado las seis metodologías ágiles encontradas.
Este conjunto de tareas cumplen ampliamente las expectativas del objetivo marcado.

3. *Identificar un conjunto de criterios para la comparación de las metodologías.*

Para la consecución de este objetivo hemos determinado las necesidades de las metodologías, buscado los factores de éxito de los proyectos Java EE, determinado los parámetros que indican la calidad del software y buscado comparativas existentes de metodologías ágiles que nos ayudasen en la realización de la comparativa .

4. *Realizar comparativa metodologías ágiles seleccionadas.*

Este objetivo, que coincide exactamente con el capítulo 6 de este documento, ha consistido en la realización de la comparativa en función de los parámetros establecidos para ello, en el objetivo anterior. Por tanto este objetivo también se puede dar por satisfecho completamente.

5. *Caso práctico de aplicación.*

Las tareas que hemos desarrollado en este objetivo han consistido en la elección de las metodologías ágiles Scrum y XP, que íbamos a utilizar para desarrollar una aplicación simple que muestra los datos recogidos en la selección de metodologías ágiles. Se ha desarrollado el producto, generado los documentos y actas requeridas y documentado la experiencia y conclusiones obtenidas. Por tanto este objetivo también esta cubierto.

6. *Identificación de herramientas necesarias en el desarrollo del software*

El último objetivo que presentamos y realizamos, ha sido satisfecho a través de la búsqueda de diferentes clasificaciones de herramientas, la elección de la mejor considerada y la identificación de los diferentes grupos de herramientas, basándonos en la clasificación encontrada. También hemos

realizado un breve estudio de las herramientas necesarias para la utilización de las metodologías ágiles y la tecnologías Java EE para el desarrollo del software.

Inicialmente planteamos algunos objetivos muy interesantes y que nos hubiera gustado poder realizar como son :

1. Selección de herramientas para el desarrollo de proyectos Java EE y su posterior prueba de producto y realización de comparativa.

2. Estudio de la metodología empresarial COM y la propuesta de mejoras y/o adaptación de métodos y técnicas ágiles para su utilización en los procesos de la empresa.

Estos últimos objetivos quedan abiertos para futuros trabajos y extensiones del proyecto realizado.

Que he aprendido

Durante seis meses he estado dedicado prácticamente al ciento por ciento a la realización de este proyecto y puedo asegurar que no ha pasado un día sin que haya aprendido una cosa nueva. Siendo más específico, he aprendido detalles de la tecnología Java EE que no conocía, he observado la diversidad de aplicaciones empresariales que pueden realizarse, cuales son sus principales características y sobretodo, que es imposible abarcarlas todas. He descubierto el origen de las metodologías del software, como surgió el concepto de metodología tradicionales, que Dijkstra supuso una revolución también para el mundo del desarrollo del software y como las metodologías han evolucionado según las necesidades de cada tiempo. He conocido nuevas metodologías que no había oído ni hablar, he podido entablar conversaciones con algunos de sus creadores o partidarios, he descubierto grupos de noticia sobre metodologías ágiles[1], que han conseguido por primera vez en mucho tiempo, que el trabajo se convirtiese en un hobby para mi. El número de bookmarks o favoritos de mi navegador se ha visto modificado e incrementado desde el inicio del proyecto, ahora sigo incondicionalmente un número de blogs en español e ingles y otras fuentes de información[2]. He aprendido a buscar libros por todos los medios posibles, ya que muchas veces la única fuente de una metodología era el libro que había escrito su autor y servicios como safaribooks [3] han sido muy útiles.
Quiero marcar un punto y aparte para comentar los conocimientos que he adquirido sobre las metodologías ágiles. Antes de iniciar este proyecto, mi visión sobre las metodologías ágiles se reducía a Extreme programing y algunas de sus técnicas más famosas o estrafalarias, como puede ser pair programming o el concepto de el desarrollo orientado al código, obviando gran parte de la documentación, si que tenia conocimientos mas o menos profundos de metodologías tradicionales y de RUP, por lo que no me sorprendía su manera de proceder, pero que una metodología como Scrum que podríamos decir que es una de las metodologías ágiles que gozan de mayor popularidad (ver selección de metodologías, capítulo 5), hubiese pasado inadvertida para mi, resultó toda una revelación de mi ignorancia. También he descubierto un mercado relacionado con la consultoría, formación e implantación de metodologías que implica a un gran número de personas, donde muchos proclaman métodos y técnicas como las definitivas, que solucionarán todos los problemas relacionados con el desarrollo del software, al menos, creo que ese espíritu crítico también lo he alimentado durante estos seis meses. Pero no solo he aprendido a criticar y discernir que parte es fanatismo y que parte es coherentemente razonable de las metodologías ágiles, sino que he llegado a la conclusión de que son una revolución, una tesis o antítesis más, y que dentro de poco aparecerá otra y luego otra, es nuestra

1 http://groups.yahoo.com/ como APM, Scrum o ExtremeProgramming son algunos buenos grupos sobre metodologías ágiles.

2 www.infoq.com, www.theserverside.com, www.implementingscrum.com, www.joelonsoftware.com, www.javahispano.com, www.ddj.com, www.navegapolis.net, www.presionblogosferica.com, etc. .

3 http://safari.oreilly.com/

manera de evolucionar, mejorar. No podemos negar que muchas técnicas y métodos de las metodologías ágiles son un avance en el desarrollo del software, con otras debemos ir con más cuidado.
Finalmente he conocido una amplia variedad de herramientas para el desarrollo del software, ya sea con metodologías ágiles, tecnología Java EE o en general, para cualquier tecnología y/o metodología. He puesto en práctica dos metodología, pudiendo observar prácticamente como se aplican y "jugar" un poco con las técnicas que promulgan.

Valoración personal

A nivel personal considero la experiencia muy positiva e inesperada por mi parte, he disfrutado realizando este documento, al mismo tiempo que iba descubriendo un mundo del cual poco tiempo atrás yo era completamente ajeno y que a día de hoy me fascina. Valoro mucho el trabajo que están realizando personas como Jim Higshmith, Alistar Cockburn, Jeff Sutherland, Kniberg, Ken Schwaber, etc, ya que a través de su experiencia y conocimientos, intentan hacer un poco más fácil el complejo proceso de desarrollo del software.
Todos los conocimientos y experiencias que he adquirido a lo largo de este proyecto me acompañarán a lo largo de mi vida profesional, permitiéndome tener un espíritu crítico frente a charlatanes metodologistas (que también los hay) y ver el mundo del software desde otra perspectiva diferente a la que tenia antes de iniciar el proyecto. Mi perspectiva inicial era prácticamente la del que cree que las cosas se tienen que hacer de una manera, la mejor, ahora entiendo que en el desarrollo del software, se pueden utilizar muchas técnicas y métodos diferentes para una misma solución, y que no existe una mejor elección para cada proyecto, sino muchas y que estas dependen de las personas que estén involucradas en el desarrollo.

Conocimientos previos aplicados

Ha sido imprescindible el conocimiento de la lengua inglesa, ya que la mayoría de documentos que he leído estaban en inglés. También han resultado muy útiles los conocimientos adquiridos a lo largo de mis años de estudiante, en asignaturas de la carrera de ingeniería informática y el máster en tecnologías de la información. Asignaturas como ingeniería del software, son básicas para poder entender los conceptos que se presentan en las metodologías, qué es el lenguaje unificado de modelado, qué son los casos de usos, qué es el ciclo de vida del desarrollo del software, ... Otras asignaturas que han sido útiles en el desarrollo de este proyecto han sido las asignaturas de PESBD, en la cual se lleva a cabo un proyecto utilizando la metodología de desarrollo RUP, las asignatura de bases de datos FBD y DABD, que me han ayudado para la realización de la base de datos de proyecto piloto y no quisiera dejarme una asignatura que me enseñó un principio tan importante como el de incertidumbre de los requisitos, Ingeniería de Requisitos (ER). En último lugar, nombro una asignatura que inicialmente no le di mucha importancia, pero que ha resultado básica, HITI, a través de la cual conocí las bases documentales que tenemos accesos a través de la facultad y que han sido la fuente de la gran mayoría de documentos científicos a los cuales se hacen referencia en este documento.
También han sido útiles conocimientos autodidactas varios o que a día de hoy podríamos llamar de cultura general o cultura informática.

11. Anexos

En este capítulo incluimos aquella documentación o documentos que creemos dar un valor añadido al proyecto, pero que dificultarían su lectura. Entre ellos podemos encontrar los detalles de los resultados de la selección de metodologías ágiles, un documento que nos envió Jim Highsmith para la realización de este proyecto, las actas de las reuniones del proyecto piloto, en definitiva, aquellos detalles que creemos necesarios incluir en el proyecto, pero que por su extensión o contenido hemos preferido ubicarlos al final del documento.

Contenidos de la sección

ANEXO 1

Criterios de selección considerados

A la hora de determinar los criterios de selección se consideraron los siguientes:

1. Moda. Número de documentos, papers y libros de texto en los últimos tres años, que indiquen la presencia en la red.
2. Mayor número de proyectos realizados y documentados.
3. Mayor número de proyectos realizados y documentados con éxito.
4. Novedad o Edad. La madurez de la metodología, ¿Cuánto tiempo hace que ha surgido?.
5. Mejor crítica o mas recomendado.
6. La metodología mas presente en Internet.
7. La metodología con una mayor comunidad de seguidores.
8. La metodología con una mayor comunidad de apoyo o mejor "soporte" (documentación, ayuda, foros, ejemplos, ….)
9. Certificaciones, training…

Clasificación metodologías

La clasificación de metodologías con mayor presencia en Internet ha sido realizada de la siguiente manera:

- Se han identificado las 5 metodologías con mayor número de búsquedas en cada uno de los buscadores.
- Se han seleccionado las metodologías coincidentes o que eran mayoría en esa posición, para cada buscador.

	1	2	3	4	5
Google	SCRUM (3420000)	XP (1190000)	TDD(492000)	CM(244000)	APM(170000)
Yahoo	SCRUM (5120000)	XP(4470000)	CM(2930000)	TDD(2800000)	APM(766000)
Live	SCRUM(1970000)	XP (1470000)	TDD(1040000)	CM(724000)	AM(538000)

tabla 1

En la posición tercera se ha escogido Test Driven Develoment (TDD), aunque el número de resultados presentados por Yahoo es superior a que la suma de los resultados obtenidos por Live y Google, primamos la coincidencia de máximos. Es decir que es la tercer metodología en cuanto resultados obtenidos en los dos otros dos buscadores mayoritarios. De la misma manera hemos seleccionado en quinta posición a Agile Project Management, por ser la quinta en dos de los tres buscadores.

La clasificación de metodologías mejor documentadas se ha realizado teniendo en cuenta el número de papers y libros encontrados.

	1	2	3	4	5
Libros	XP (22)	TDD(9)	DSDM(6)	SCRUM(6)	El resto(1 o 0)
Papers	XP (100)	DSDM(70)	TDD(55)	SCRUM(43)	APM(8)

tabla 2

En libros hemos sumado libros en español y otros idiomas (en inglés). La clasificación ha sido realizada de la siguiente forma:

- Cuando aparece la misma metodología en la misma posición para Libros y papers, escogemos la metodología en esa posición.
- Sino aparece la metodología en la misma posición priorizamos el número de libros.
- En el caso de la quinta posición de los libros escogemos Agile Project Management, ya que es la metodología que tiene mayor número de papers y un libro publicado.

Tendencias de las metodologías ágiles

No hemos incluido ni tenido en cuenta este parámetro para la selección de las metodologías, pero hemos considerado interesante incluirlo como anexo y punto de interés para el lector.
Según la página http://www.indeed.com/, que realiza búsquedas de empleo, la tendencia de las metodologías según el porcentaje de búsquedas de trabajo, relacionadas con ellas, que se han llevado a cabo es la siguiente:

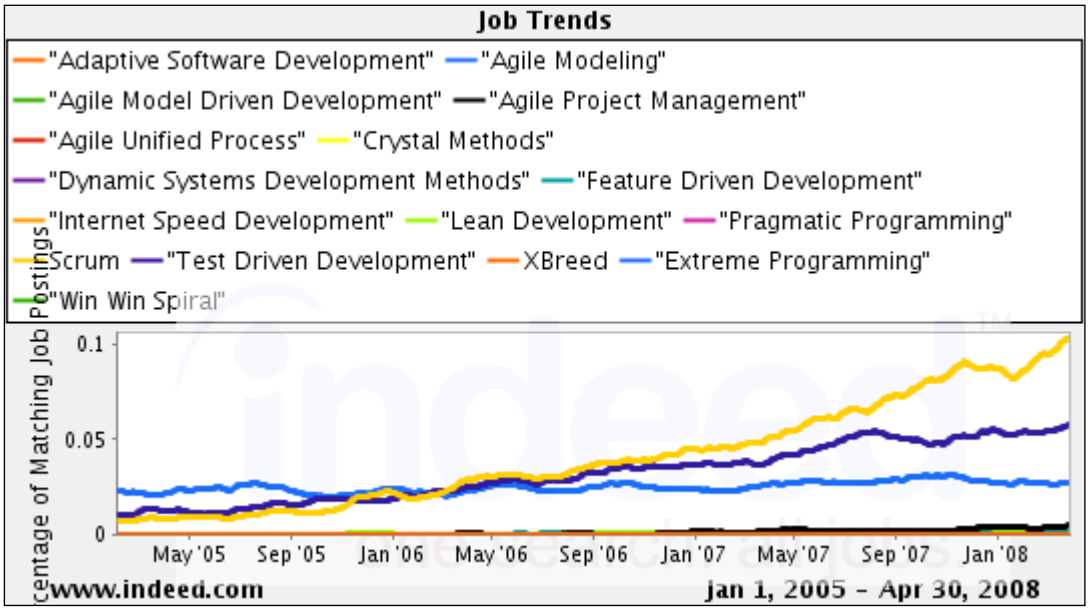


fig 1

Destacan Scrum, Test Driven Development y Extreme Programming. A pesar de que no se observe en el gráfico, Agile Project Management también ha experimentado una tendencia creciente.

Resultados detallados la selección

1. *Adaptive Software Development*

- Búsqueda Web,
  - Clave de la búsqueda: "Adaptive Software Development"

- Búsqueda de Libros,
  - Clave de la búsqueda: "Adaptive Software Development"
  - Libros encontrados:
    - 1. Adaptive Software Development: A collaborative Approach to Managing Complex Systems. James A.Highsmith III. English. Dorset House Publishing Company, 1999.

- Búsqueda de Papers,
  - Clave de la búsqueda: "Adaptive Software Development"
  - 21 resultados obtenidos de los cuales, 2 relevantes:
    - Engineering Village, 2 nuevos.
    - IEEExplore, 1 repetido.
    - Science Direct,0 relevantes.
  - Papers encontrados :
    - 1. An adaptive Software development process model.
    - 2. Messy, exciting, and anxiety-ridden adaptive software development.

2. *Agile Modeling*

- Búsqueda Web,
  - Clave de la búsqueda: "Agile Modeling"
- Búsqueda de Libros,
  - Clave de la búsqueda: "Agile Modeling"
  - Libros encontrados:
    - 1. Agile Modeling: Effective Practices for Extreme Programming and the Unified Process. Scott W. Ambler and Ron Jeffries. Wiley, 1st Edition (March 22, 2002).

- Búsqueda de Papers,
  - Clave de la búsqueda: "Agile Modeling"
  - 6 resultados obtenidos :
    - Engineering Village, 4
    - IEEExplore, 2
    - ScienceDirect, 0

3. *3-Agile Model Driven Development*

- Búsqueda Web,
  - Clave de la búsqueda: "Agile Model Driven Development"
- Búsqueda de Libros
  - Libros encontrados:

- 1. The Object Primer: Agile Model-Driven Development with UML 2.0. Scott W.Ambler. English. Cambridge University Press; 3th Edition 2004.

- Búsqueda de Papers
  - Clave de la búsqueda: "Agile Model Driven Development"
  - 2 resultados obtenidos:
    - Engineering Village, 1
      •Agile model driven development is good enough.
    - IEEExplore, 1
      •Facilitating Agile Model Driven Development and End-user Development for evolving Web based Workflow applications.
    - Science Direct, 0.

4. *Agile Project Management*

- Búsqueda Web
  - Clave de la búsqueda: "Agile Project Management"
- Búsqueda de Libros
  - Clave de la búsqueda: "Agile Project Management"
  - Libros encontrados :
    - 1. Agile Project Management:Creating Innovative Products (The agile software development series) Jim Highsmith. Addison-Wesley Professional (April 16, 2004)

- Búsqueda de Papers
  - Clave de la búsqueda: "Agile Project Management"
  - Se han encontrado 8 papers con la palabra clave en el título, en las bases documentales IEEExplore y Engineering Village..

5. *Agile Unified Process*

- Búsqueda Web
  - Clave de la búsqueda: "Agile Unified Process"
- Búsqueda de Libros
  - Clave de la búsqueda: "Agile Unified Process"
  - Libros encontrados: ninguno.
- Búsqueda de Papers
  - Clave de la búsqueda: "Agile Unified Process"
  - Resultados no relevantes.

6. *Crystal Methods*

- Búsqueda Web
  - Clave de la búsqueda:
    - Cristal AND (Clear OR Yellow OR Orange OR "Orange Web" OR Red OR Maroon) AND methodology
- Búsqueda de Libros:
  - Clave de la búsqueda: "Crystal methods"
  - Libros encontrados:

- 1. Crystal clear: A Human-Powered Methodology for Small Teams. Alistair Cockburn. Addison-Wesley Professional, 1st Edition (2004).

* Búsqueda de Papers
  - Ningún resultado relevante encontrado.
    - Engineering Village, 3. Referencias a la metodología, nunca como eje central del paper.
    - IEEExplore, 13 papers que mencionan la familia de metodologías crystal.
    - Science Direct, 0 referencias.

7. *Dynamic Systems Development Method*

* Búsqueda Web
  - Clave de la búsqueda: "Dynamic Systems Development Method"
* Búsqueda de Libros
  - Clave de la búsqueda: "Dynamic Systems Development Method"
  - Libros encontrados (6):
    - 1. Dynamic Systems Development Method. Jennifer Ann Margaret Stapleton. Tesseract Publishing, 2rev Edition 1996.
    - 2. DSDM:Business Focused Development, Second Edition. DSDM Consortium, Jennifer Stapleton. Pearson Education, 2 edition 2003.
    - 3. Agile Project management: running PRINCE2 projects with DSDM Atern. Keith Richards 2007.
    - 4. DSDM:Business Focused Development, 2nd Edition 2003.
    - 5. The DSDM Consortium. 2003.
    - 6. The DSDM Student workbook, 2002.
* Búsqueda de Papers
  - Clave de la búsqueda:
    - "Dynamic Systems Development Method" OR DSDM
  - 70 resultados obtenidos:
    - Engineering Village, 70.
    - IEEEXplore, 3 resultados incluídos en la búsqueda de EV.
    - Science Direct, 0.

8. *Feature Driven Development*

* Búsqueda Web
  - Clave de la búsqueda: "Feature Driven Development"
* Búsqueda de Libros
  - Clave de la búsqueda: "Feature Driven Development" OR FDD
  - Libros encontrados:
    - 1. A practical guide to Feature-driven Development. Stephen R.Palmer, John M.felsing, Steve Palm. Prentice Hall P/R 2002..

* Búsqueda de Papers
  - Clave de la búsqueda: "Feature Driven Development"
  - 3 resultado relevante obtenidos:
    - Engineering Village, 14 resultados, de los cuales sólo 3 consideran FDD como núcleo del paper.
    - IEEExplore, 1 paper y poco relevante.
    - Science Direct, 0.

9. *Internet Speed Development*

- Búsqueda Web
  - Clave de la búsqueda: "Internet Speed Development"
- Búsqueda de Libros
  - Clave de la búsqueda: "Internet Speed Development"
  - No se han encontrado libros específicos de la metodología, sin embargo es mencionado en diferentes libros:
    1. Advanced Information System Engineering: 19th Internacional Conference.
    2. The past and the future of Information Systems
    3. Mirando alrededor. El día a día de los proyectos software.
- Búsqueda de Papers
  - No se han encontrado resultados relevantes, solo se menciona la metodología en 2 papers.

10. *Lean Development*

- Búsqueda Web
  - Clave de la búsqueda: "Lean Development" methodology
- Búsqueda de Libros
  - No se han encontrado libros específicos de la metodologías, sin embargo le dedican un capítulo en:
    1. Agile Software Development Ecosystems. Addison Wesley 2002. English. Jim Highsmith, James A.Highsmith. "Chapter 21"
- Búsqueda de Papers
  - 2 resultados relevantes:
    - Engineering Village, 15 resultados, de loc cuales solo uno es relevante.
    - IEEExplore, 3 resultados de los cuales 1 relevante:
    - Incorporating Lean Development Practices into Agile Software Development.
    - Science Direct, 0.

11. *Pragmatic Programming*

- Búsqueda Web
  - Clave de la búsqueda: "Pragmatic Programming"
- Búsqueda de Libros
  - Clave de la búsqueda: "Pragmatic Programming"
  - Ningún libro encontrado.
- Búsqueda de Papers
  - Clave de la búsqueda: "Pragmatic Programming"
  - Ningún paper encontrado con esta metodología.

12. *Scrum*

- Búsqueda Web
  - Clave de la búsqueda: "SCRUM" development
- Búsqueda de Libros
  - Se han encontrado 2 libros en castellano y 4 en ingles, siendo uno de los libros en castellano una traducción.

1. Flexiilida con Scrum. Juan Palacio. 2007/ 2008. SafeCreative
2. Agile Project management with Scrum. Ken Schwaber.
3. Agile Software development with Scrum. Ken Schwaber.
4. The Enterprise and Scrum. Allan Gerald.
5. Scrum and XP from the Trenches.
6. Versión en castellano, Scrum y XP desde las trincheras.

- Búsqueda de Papers
  - Se han encontrado 43 papers que incluyen en su título el término SCRUM, en referencia a la metodología de desarrollo. Estos resultados se han obtenido de la búsqueda en las bases documentales de IEEExplore y Engineering Village.

13. Test Driven Development

- Búsqueda Web
  - Clave de la búsqueda: "Test Driven Development"
- Búsqueda de Libros
  - Clave de la búsqueda: "Test Driven Development"
  - Se han encontrado 9 libros:
    1. Test Driven Development: By Example. Kent Beck
    2. Test Driven Development in Microsoft.Net. James W.Newkirk.
    3. Mock Objects and Test Driven Development. Steve Freeman.
    4. Test Driven Development: A practical guide. David Astek
    5. Test Driven: TDD and Acceptance TDD for Java Developers.
    6. …

  - Búsqueda de Papers
    - Clave de la búsqueda: "Test Driven Development"
    - Se han obtenido 55 resultados, 19 de IEEEXplore incluídos en los 55 de Engineering Village.

14. XBreed

- Búsqueda Web
  - Clave de la búsqueda: Xbreed Software
- Búsqueda de Libros
  - Clave de la búsqueda: Xbreed
  - No se han encontrado libros específicos de esta metodología.
- Búsqueda de Papers
  - Clave de la búsqueda: Xbreed
  - No se han encontrado ningún paper relevante que mencione esta metodología.

15. Extreme Programming

- Búsqueda Web
  - Clave de la búsqueda: "Extreme Programming"
- Búsqueda de Libros
  - Clave de la búsqueda: "Extreme Programming"
  - Se han encontrado 2 libros en castellano y más de veinte libros en ingles. Destacamos la serie "The XP series", de la cual es autora Kent Beck.
- Búsqueda de Papers
  - Clave de la búsqueda: "Extreme Programming"

- Se han encontrado mas de 100 que incluyen en su título el término "Extreme Programming", en referencia a la metodología de desarrollo. Estos resultados se han obtenido de la búsqueda en las bases documentales de IEEExplore y Engineering Village.

16. *Win Win Spiral*

- Búsqueda Web
  - Clave de la búsqueda: "Win Win Spira Model"
- Búsqueda de Libros
  - Clave de la búsqueda: "Win Win Spira Model"
  - No se han encontrado libros específicos de esta metodología.
- Búsqueda de Papers
  - Se han encontrado 3 papers en las bases documentales de IEEExplore y Engineering Village.

ANEXO 2

Documento Highsmith.

A BASELINE AGILE FRAMEWORK (DRAFT)

A Baseline Agile Framework

This instrument addresses the core principles and practices that all agile teams should consider. Most of the practices are drawn from Agile Project Management (Highsmith), Scrum (Schwaber), and XP (particularly Industrial XP from Josh Kerievsky). Additional practices or ideas have been drawn from other agile sources.

The first part of this instrument is an evaluation of agile principles and how they are implemented, or to be implemented, in your organization. Each statement should be evaluated by indicating its importance in your organization on a 1-5 scale: 1-not important, 2-a bit important, 3-somewhat important, 4-very important, 5-extremely important.

Each agile practice statement should be evaluated on a 1-5 scale: 1-do not plan to implement, 2-may implement sometime, 3-plan to implement sometime, 4-plan to implement next wave, 5-plan to implement now.

Key Principles of Agile Project Management & Development

The key principles of agile project management and agile software development are embedded in the Agile Manifesto (software) (http://www.agilemanifesto.org/) and the Declaration of Interdependence (project management) (http://pmdoi.org/).

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done—is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.

- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Practices—Getting Agile Projects Started and Planned

- Readiness Assessment (Are teams ready for an agile project/process?)
- Product & Project Vision
  - Vision (Box & Elevator Statement)
  - Product Architecture (Skeleton)
  - Project Data Sheet (Scope & Boundaries)
  - Business Objectives (test-driven management)
  - Risk Assessment
- People & Process Vision
  - Project Community Definition
  - Customer Team—Developer Team Roles & Responsibilities, Product Owner/Manager, Project Leader
  - Process & Practice Tailoring
  - Team Values & Working Agreements
- Release Planning
  - Create the product backlog items (Stories)
  - Develop the release plan
- Chartering (Collaborative session to create the vision & planning deliverables)

Agile Practices—Practices to Deliver Results

- Project Management
  - Coaching & Team Development
  - Team workload management
  - Participatory Decision Making
  - Sustainable Pace
- Technical/Product
  - Story Development
  - Iteration Planning
  - Evolutionary design
  - Refactoring
  - Pattern or Domain-driven design
  - Test-driven Development
  - Pairing (or other small group collaboration practice)

- Frequent builds (at least 1/day)
- Continuous Integration
- Coding standards
- Automated testing (to what degree are story, unit, and QA testing automated?)

➢ Collaboration

- High level of Customer/Product Management involvement
- Daily Stand-up or Scrum Meetings
- Collocation
- Collective Ownership
- Daily interaction with the customer team
- Distributed teams (virtual collocation practices & tools)

Agile Practices—Reflect, Learn, and Adapt

➢ Customer Acceptance

- Customer Focus Groups (demo)
- Storytesting

➢ Retrospectives (iteration & project)

➢ Continuous Learning

➢ Project Reporting

➢ Adaptive Action Planning

ANEXO 3

Indicaciones

Tools Taxonomy

Tool Name - Name of tool classified.

Version Identification/Release Date - Version number and release date of the tool.

Tool Supplier - Vendor supplying the tool, including address and phone number.

Brief Description - One- or two-sentence description of tool.

Development Phase / ***Operation on Object***	***Project Management***	***System/SW Req'ts Analysis***	***SW Requirements Analysis***	***Preliminary Design***	***Detailed Design***	***Coding and Unit Testing***	***CSC Integration and Testing***	***CSCI Testing***	***Sys Integ/Test***	***Other***
create										
transform										
group										
analyze										
refine										
import										
export										
other										

Special Definitions

Place X's in the boxes of the taxonomy to indicate tool functionality in a phase of the development cycle.

Definitions for the row and column labeled "other" should be included here if they are used.

Objects and Operations

For each object the tool deals with, describe the object and each of the operations performed on the object.

Attributes

- ***Required Computer/OS -*** Hardware and operating system (if needed) for tool operation
- ***Other Required Hardware -*** Any other required hardware such as graphics monitors, mouse, or bit pads
- ***Other Required Software -*** Any other required software such as a spreadsheet program or database manager
- ***Methodology Supported -*** Methodology(s), if any, supported by the tool
- ***Other Tools Interfaces -*** Other tools that can be integrated or used in conjunction with the tool
- ***Price Range -*** Price ranges are: Low (up to $10K); Medium ($10 to $30K); High (above $30K). Ranges were chosen as a means to represent the cost of one copy of the tool as prices vary relative to such variables as the buyer, supplier, and quantity purchased.

Comments

This section should include other information not reflected by the taxonomy and the objects and operations descriptions.

- Special features that are especially noteworthy
- Comments or notes on user interfaces
- Constraints on the tool, for example, works only for the Ada programming language
- Tool set structure (what components comprise the tool)
- Basis for the tool classification: vendor supplied, tool user, review of brochures, or product documentation
- Tool history
- Other.

Ejemplos

Example #1 - Tools Taxonomy

Tool Name - Computer-Aided Software Engineering Environment (CASEE)/Project Manager

Version ID/Release Date -
Ver 1.5, Jan '87 release date

Tool Supplier - Software Engineering Corporation of Tomorrow
1111 Somewhere Ave.
Anyplace, USA 00001

Brief Description -
Project Manager is part of an integrated software tool set, CASEE. It provides a database and a set of functions to support project planning, scheduling, estimating, and management.

Development Phase / ***Operation on Object***	***Project Management***	***System/SW Req'ts Analysis***	***SW Requirements Analysis***	***Preliminary Design***	***Detailed Design***	***Coding and Unit Testing***	***CSC Integration and Testing***	***CSCI Testing***	***Sys Integ/Test***	***Other***
create	X									
transform	X									
group										
analyze	X									
refine	X									
import										
export	X									
other										

Special Definitions

N/A

Objects and Operations

Project Management

1. *Project and Task Descriptions*
    a. *creates* project and task descriptions as textual objects with defined fields for priority, precedence, and time estimate for completion
    b. *transforms* project/task descriptions and resources against a time scale automatically into a Gantt chart
    c. *transforms* tasks, priority, precedence, and time estimate for completion automatically into a Pert chart
    d. *transforms* project tasks and expended resources into status reports
    e. *refines* project and task descriptions into subprojects or subtasks
    f. *exports* descriptions in predefined report format to printer/plotter
2. *Resource Descriptions*
    a. *creates* resource descriptions such as manpower or equipment
    b. *exports* resource descriptions in predefined report format to printer/plotter
3. *Calendar*
    a. *creates* pictorial calendar with flexibility to distinguish holidays, weekends, and working days
    b. *refines* calendar entries with time resolution to the hour or minute
    c. *exports* calendar to printer/plotter
4. *Gantt chart*
    a. *creates* Gantt chart through graphic interface as well as automatically
    b. *exports* Gantt chart to printer/plotter
5. *Pert chart*
    a. *analyzes* Pert chart by determining critical path(s) of project tasks, "what if" reports when tasks and/or resources are changed, and time required to complete each task
    b. *exports* Pert chart to printer/plotter
6. *Status Reports*
    a. *analyzes* interrelationships between resources
    b. *exports* status reports via predefined report template

Attributes

- ***Required Computer/OS -*** MicroVAX/VMS, VAX/VMS[1]
- ***Other Required Hardware -*** TEK 4111[2] (1 Mbyte user buffer), mouse, printer/plotter
- ***Other Required Software -***
- ***Methodology Supported -*** Tailorable to user's own project management methodology

- ***Other Tools Interfaces -*** Interfaces to CASEE Environment (Analyzer, Designer, Code Generator); export utility allows reports to be prepared with other word processing and spreadsheet packages
- ***Price Range -*** Low (less than $10K)

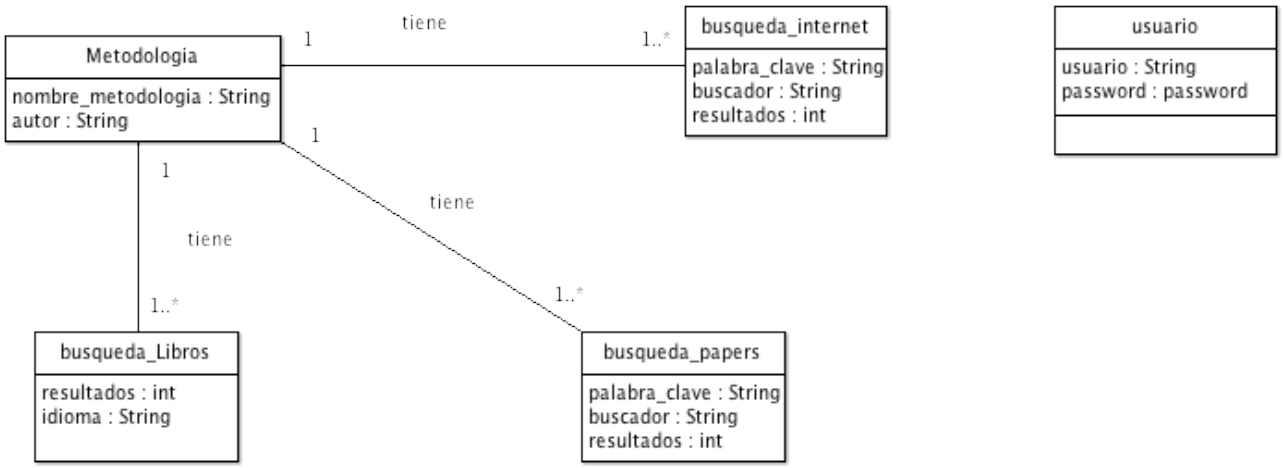
Comments

- Allows user to handle unlimited numbers of tasks and projects
- Allows graphical and textual input
- Allows personal calendar appointments/entries
- Limitation: Once tasks are defined they cannot be regrouped (they must be deleted and reentered)
- Limitation: Can only assign a maximum of six resources to a single task; resources cannot be shared across tasks
- Limitation: Does not assist with budget and cost accounting
- Basis for tool classification: Perusal of vendor brochures and user's manuals
- CASEE tool set has been on the market since July '80, has sold over 100 copies, and is in use in approximately 15 companies. The Project Manager is a newer extension of the tool set and has been released since Feb '86.

ANEXO 4

Modelo conceptual base de datos.

Una de las tareas que debían desarrollarse en el proyecto piloto era la creación del modelo conceptual de la base de datos. Es una técnica esencial para poder generar una base de datos, sobretodo en aplicaciones tan sencillas que no especificamos prácticamente ningún otro modelo de datos.



metodologia (nombre, autor, año)

internet (pclave, buscador , resultados, fecha)

busquedapaper (pclave, buscador, resultados, fecha)

metodologiabusquedalibros (nombre_metodologia, resultados, idioma, fecha)
clave foránea nombre_metodología de tabla metodología(nombre).

metodologiapapers (nombre_metodologia, pclave_papers, numeropapers)
clave foránea pclave_papers de tabla busquedapaper(pclave).
clave foránea nombre_metodología de tabla metodología(nombre).

metodologiainternet (nombre_metodologia, pclave_internet)
clave foránea nombre_metodología de tabla metodología(nombre).
clave foránea pclave_internet de tabla internet(pclave).

usuarios (usuario, password)

Reuniones

A continuación incluimos las actas de las reuniones desarrolladas en el proyecto piloto, las cuales siguen la plantilla de actas de reunión las asignatura de Enginyeria de Requisits de la FIB.

1. *Acta de la reunión de planificación del primer sprint.*

Piloto Metodologías Ágiles

Anotaciones de la reunión

Equipo: 1 Día: 22/05/08 Inicio/Fin hora: 08h30/14h Oficinas de Everis Diagonal

Asistentes:

Jose Carlos Carvajal (Scrum Master y Product Owner)

Jose Carlos Carvajal (Equipo de desarrolladores.)

Ausentes:

No hay ausentes.

Temas a tratar en la reunión:

1-Planificación del Sprint 1

A continuación especificamos parámetros del sprint:

Fijamos una duración del sprint de 15 días laborables (3 semanas).

Estimación de la velocidad mediante un cálculo de recursos:

Equipo de 1 persona, disponible al 50%.

Días Disponibles

Jose Carlos 15

Total = 15 días-hombre disponibles

La velocidad estimada es inferior a 15, veamos cuanto inferior, utilizamos “factor de dedicación”:

(DÍAS-HOMBRE DISPONIBLE) X (FACTOR DE DEDICACIÓN) = VELOCIDAD ESTIMADA

15 x 0.5 = 7.5 (velocidad estimada)

Selección de las historias del product backlog:

PILA DE PRODUCTO:

Item Id	Nombre de la historia	Estado	Estimación	Sprint	Prioridad	Comentarios
1	Diseño apariencia de la aplicación.	Plan.	8	1	1	Modelo conceptual de la BBDD e insertar datos.
2	Mostrar comparativa del estudio de Metodologías.	Plan.	6	2	2	Utilización de servlets, jsp. Contenido dinámico
3	Login de usuario.	Plan.	3	3	3	
4	Añadir nuevas metodologías al estudio.	Plan.	3	3	4	Verificar campos obligatorios y formato de los datos introducidos.
5	Eliminar metodologías y todos sus datos.	Plan.	2	3	5	
6	Modificación datos introducidos.	Plan.	3	-	6	
7	Añadir papers como resultado de una búsqueda.	Plan.	6	-	7	Como se suben ficheros al servidor.
8	Consultar palabras clave de las búsquedas.	Plan.	1	-	8	
9	Top 5 metodologías ágiles.	Plan.	5	-	9	
10	Realizar automáticamente búsquedas y actualizar los datos.	Plan.	10	-	10	
11	Generar gráficos mejor documentación y mayor presencia en Internet.	Plan.	10	-	11	

Seleccionamos las primera historia de la pila, Diseño de la apariencia de la aplicación.

Planificación del Sprint 1, tareas:

Item Id:	**1**	Item Backlog:	Diseño apariencia aplicación.
Tarea:	Preparar entorno y diseño de la gui.		
Responsable: Jose Carlos			
Estimación inicial	**1**	Trabajo hecho 0	Trabajo restante 1

Item Id:	**1**	Item Backlog:	Diseño apariencia aplicación.
Tarea:	Introducción datos BBDD		
Responsable: Jose Carlos			
Estimación inicial	**2**	Trabajo hecho 0	Trabajo hecho 2

Item Id:	**1**	Item Backlog:	Diseño apariencia aplicación.
Tarea:	Diseño GUI y disposición datos		
Responsable: Jose Carlos			
Estimación inicial	**2**	Trabajo hecho 0	Trabajo restante 2

Item Id:	**1**	Item Backlog:	Diseño apariencia aplicación.
Tarea:	Preparación entorno		
Responsable: Jose Carlos			
Estimación inicial	**1**	Trabajo hecho 0	Trabajo restante 3

2- Herramienta gestión de las tareas Scrum

Inicialmente se utilizó la herramienta IceScrum. Es una herramienta atractiva, funcional y open source que permite la creación del plan del producto, la planificación de sprints, representación de roles y generación automática de gráficos de seguimiento (burn-down) , pero aún no disponen de una versión lo suficientemente estable. Es una aplicación web desarrollada en Java EE y utiliza tecnologías como icefaces y spring.
Finalmente se decidió realizar el seguimiento del proyecto utilizando Excel y una plantilla bastante completa, que podemos encontrar en http://www.odd-e.com/ .

Resumen de los detalles discutidos y/o acuerdos alcanzados en la reunión

Planificación del primer Sprint y sus tareas.

Elección herramienta de gestión, Excel.

Asignación de tareas:

Jose Carlos es Product Owner.

Jose Carlos es Scrum Master.

Jose Carlos es equipo de desarrollo.

Planificación de los puntos a discutir y/o que deben realizarse para la siguiente reunión:

Demo del primer Sprint, que muestre una página html estática con los datos de la comparativa de metodologías.

Revisión de la velocidad del sprint.

Problemas impedimentos en sprint 1

Planificación del segundo sprint.

2. *Acta de la reunión final del primer sprint y planificación del segundo sprint.*

Piloto Metodologías Ágiles

Anotaciones de la reunión

Equipo: 16 Día: 12/06/08 Inicio/Fin hora: 08h30/14h Oficinas de Everis Diagonal

Asistentes:

Jose Carlos Carvajal (Scrum Master y Product Owner)

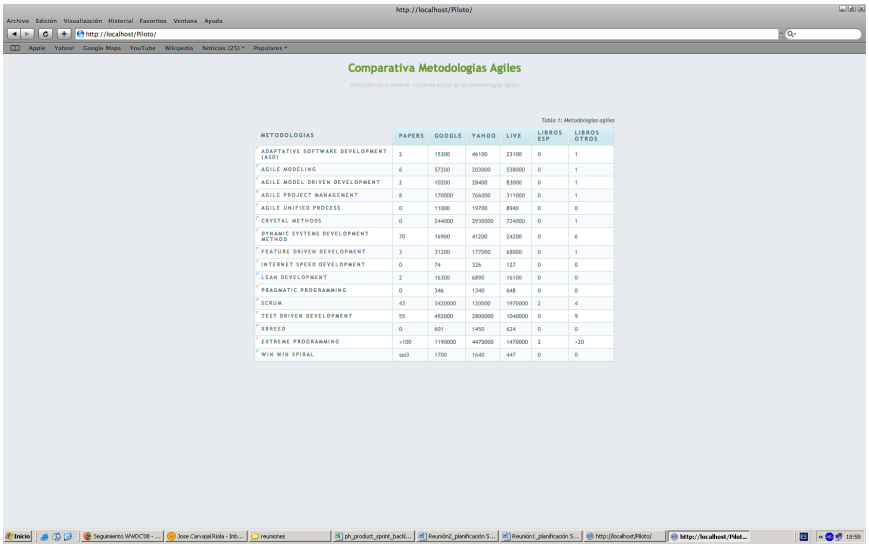
Jose Carlos Carvajal (Equipo de desarrolladores.)

Ausentes:

No hay ausentes.

Temas a tratar en la reunión:

1-Demo del Sprint1



METODOLOGIAS	PAPERS	GOOGLE	YAHOO	LIVE	LIBROS ESP	LIBROS OTROS
ADAPTATIVE SOFTWARE DEVELOPMENT (ASD)	2	15300	46100	23100	0	1
AGILE MODELING	6	57200	203000	538000	0	1
AGILE MODEL DRIVEN DEVELOPMENT	2	10200	28400	83000	0	1
AGILE PROJECT MANAGEMENT	8	170000	766000	311000	0	1
AGILE UNIFIED PROCESS	0	11000	19700	8940	0	0
CRYSTAL METHODS	0	244000	2930000	724000	0	1
DYNAMIC SYSTEMS DEVELOPMENT METHOD	70	16900	41200	24200	0	6
FEATURE DRIVEN DEVELOPMENT	3	31200	177000	68000	0	1
INTERNET SPEED DEVELOPMENT	0	74	326	127	0	0
LEAN DEVELOPMENT	2	16300	6890	16100	0	0
PRAGMATIC PROGRAMMING	0	346	1340	648	0	0
SCRUM	43	3420000	120000	1970000	2	4
TEST DRIVEN DEVELOPMENT	55	492000	2800000	1040000	0	9
XBREED	0	601	1450	624	0	0
EXTREME PROGRAMMING	+100	1190000	4470000	1470000	2	+20
WIN WIN SPIRAL	sss3	1700	1640	447	0	0

2-Problemas e impedimentos del primer Sprint

Dedicación parcial o inferior. Imprevistos temporales.

Necesidad de realizar las tareas administrativas (generación de pila de producto, tareas, etc …), consume mucho tiempo de desarrollo. Destacamos la importancia de tener roles dedicados.

No han podido realizarse todas las tareas de la historia, por lo que hemos de revisar nuestra velocidad e incluir las tareas no finalizadas en el siguiente sprint.

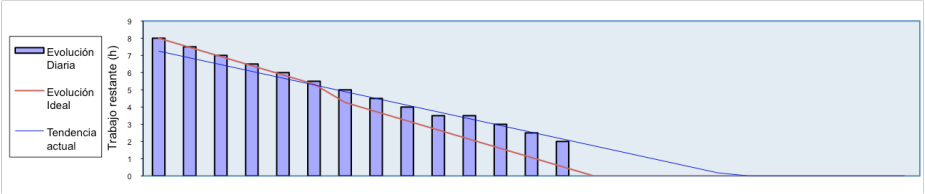
3-Revisión de la velocidad del Sprint 1

La velocidad del primer Sprint ha sido de 6 unidades de historia o días ideales.

Sprint 1 Backlog

INICIO 22/05/2008
FIN 11/06/2008

Sprint implementation days	15			Effort	Remaining on implementation day…													
Trend calculated based on last	7,5	Days	Totals	8	8	7,5	7	6,5	6	5,5	5	4,5	4	3,5	3,5	3	2,5	2
Task name	Story ID	Responsible	Status	Est.	1	2	3	4	5	6	8	9	10	11	12	13	14	15
Modelo Conceptual BBDD	1	Jose Carlos	Done	1	1	0,5	0	0	0	0	0	0	0	0	0	0	0	0
Introducción datos BBDD	1	Jose Carlos	Done	2	2	2	2	1,5	1	0,5	0	0	0	0	0	0	0	0
Diseño GUI y disposición datos	1	Jose Carlos	Done	2	2	2	2	2	2	2	2	1,5	1	0,5	0,5	0	0	0
Preparación entorno - Instalacion Máquinas virtuales	1	Jose Carlos	Done	3	3	3	3	3	3	3	3	3	3	3	3	3	2,5	2



A pesar de que la velocidad desarrollada en el primer Sprint ha sido de 6 unidades de historia, por limitaciones temporales del proyecto, debemos aumentarla como mínimo a 7,5 unidades de historia en los siguientes Sprints, incluyendo las tareas no finalizadas en el mismo. Veamos como se desarrolla el proyecto con este cambio.

4-Planificación del segundo Sprint

A continuación especificamos parámetros del sprint:

Fijamos una duración del sprint de 15 días laborables (3 semanas).

Estimación de la velocidad mediante un cálculo de recursos:

Equipo de 1 persona, disponible al 50%.

Días Disponibles

Jose Carlos 15

Total = 15 días-hombre disponibles

La velocidad estimada es inferior a 15, veamos cuanto inferior, utilizamos “factor de dedicación”:

(DÍAS-HOMBRE DISPONIBLE) X (FACTOR DE DEDICACIÓN) = VELOCIDAD ESTIMADA

15 x 0.5 = 7.5 (velocidad estimada)

A pesar de que la velocidad del anterior Sprint fue de 6, consideramos que podemos incrementar la velocidad en 1,5 puntos de historia para este Sprint.

Selección de las historias del product backlog:

PILA DE PRODUCTO:

Item Id	Nombre de la historia	Estado	Estimación	Sprint	Prioridad	Comentarios
1	Diseño apariencia de la aplicación.	Hecho	8	1	1	Modelo conceptual de la BBDD e insertar datos.
2	Mostrar comparativa del estudio de Metodologías.	Plan.	6	2	2	Utilización de servlets, jsp. Contenido dinámico
3	Login de usuario.	Plan.	3	3	3	
4	Añadir nuevas metodologías al estudio.	Plan.	3	3	4	Verificar campos obligatorios y formato de los datos introducidos.
5	Eliminar metodologías y todos sus datos.	Plan.	2	3	5	
6	Modificación datos introducidos.	Plan.	3	-	6	
7	Añadir papers como resultado de una búsqueda.	Plan.	6	-	7	Como se suben ficheros al servidor.
8	Consultar palabras clave de las búsquedas.	Plan.	1	-	8	
9	Top 5 metodologías ágiles.	Plan.	5	-	9	
10	Realizar automáticamente búsquedas y actualizar los datos.	Plan.	10	-	10	
11	Generar gráficos mejor documentación y mayor presencia en Internet.	Plan.	10	-	11	

No se modifican los ítems de la pila de producto. Seleccionamos siguiente historia significativa.

Planificación del Sprint 2, tareas:

Item Id:	**2**	Item Backlog:	Mostrar comparativa del estudio de Metodologías.		
Tarea:	Preparación entorno - IC, Subversion,…				
Responsable: Jose Carlos					
Estimación inicial	**2**	Trabajo hecho	0	Trabajo restante	2

Item Id:	**2**	Item Backlog:	Mostrar comparativa del estudio de Metodologías.		
Tarea:	Formación JSP y servlets				
Responsable: Jose Carlos					
Estimación inicial	**2**	Trabajo hecho	0	Trabajo restante	2

Item Id:	**2**	Item Backlog:	Mostrar comparativa del estudio de Metodologías.		
Tarea:	Diseñar Arquitectura de la aplicación				
Responsable: Jose Carlos					
Estimación inicial	**1**	Trabajo hecho	0	Trabajo restante	1

Item Id:	**2**	Item Backlog:	Mostrar comparativa del estudio de Metodologías.		
Tarea:	Definir y escribir pruebas				
Responsable: Jose Carlos					
Estimación inicial	**1**	Trabajo hecho	0	Trabajo restante	1

Item Id:	**2**	Item Backlog:	Mostrar comparativa del estudio de Metodologías.		
Tarea:	Obtener datos comparativa de la BBDD				
Responsable: Jose Carlos					
Estimación inicial	**1,5**	Trabajo hecho	0	Trabajo restante	1,5

Resumen de los detalles discutidos y/o acuerdos alcanzados en la reunión

Análisis del gráfico burn-down, determina que es necesario un incremento de la velocidad para alcanzar los objetivo. La velocidad debe ser de 7,5 puntos de historia.

Planificación del segundo Sprint y sus tareas.

Asignación de tareas:

Jose Carlos es Product Owner.

Jose Carlos es Scrum Master.

Jose Carlos es equipo de desarrollo.

Planificación de los puntos a discutir y/o que deben realizarse para la siguiente reunión:

Demo del segundo Sprint, que muestre la aplicación web con los datos de la comparativa de metodologías, obtenidos de los datos disponibles en la base de datos.

Revisión y análisis de la velocidad y productividad del sprint.

Señalar problemas e impedimentos en el segundo Sprint.

Planificación del tercer sprint.

3. *Acta de la reunión final del segundo sprint y planificación del tercer sprint.*

Piloto Metodologías Ágiles

Anotaciones de la reunión

Equipo: 22 Día: 04/07/08 Inicio/Fin hora: 08h30/14h Oficinas de Everis Diagonal

Asistentes:

Jose Carlos Carvajal (Scrum Master y Product Owner)

Jose Carlos Carvajal (Equipo de desarrolladores.)

Ausentes:

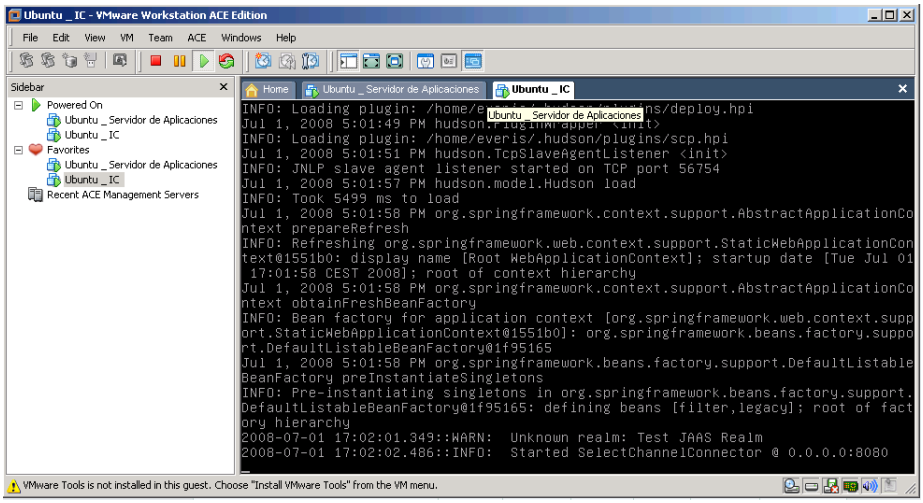
No hay ausentes.

Temas a tratar en la reunión:

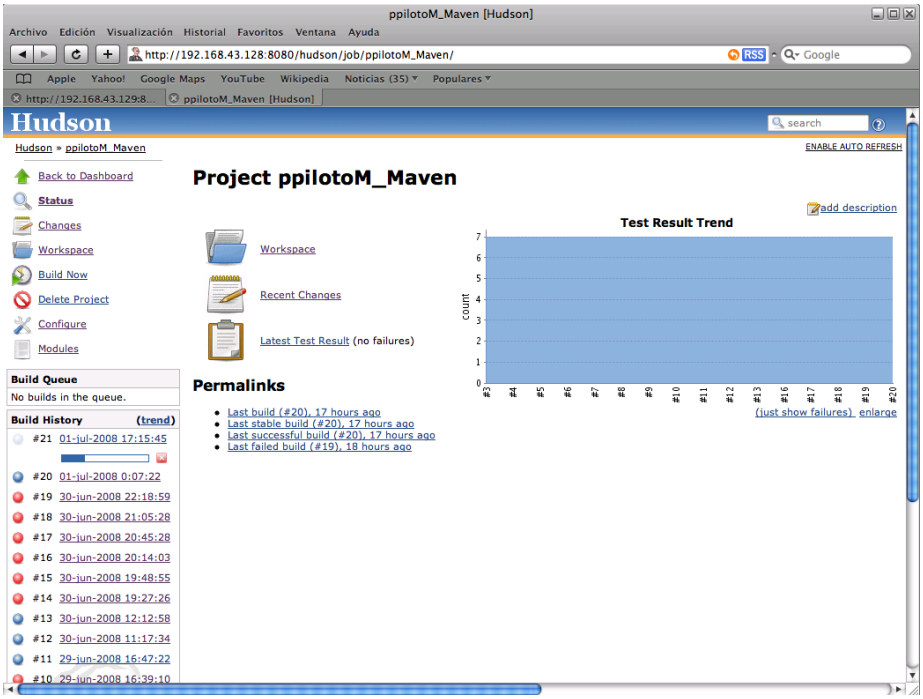
1-Demo del Sprint2

Consiste en la muestra de los resultados de las pruebas realizadas y el entorno de desarrollo en el que se han desarrollado.

Máquinas virtuales instaladas y funcionando:



Entorno de Integración Continua Hudson:



Resultados de la ejecución de las pruebas con Junit:

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.everis.maven.TestCaseMetodologia
Successfully connected to MySQL server using TCP/IP...
Successfully connected to MySQL server using TCP/IP...
Prueba metodologia vacia se crea correctamente
Prueba metodologia llena se crea correctamente
Successfully connected to MySQL server using TCP/IP...
Prueba obtener todas las metodologias de la bbdd
Successfully connected to MySQL server using TCP/IP...
Prueba obtener una metodologia inexistente de la bbdd
Tests run: 7, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 7.388
sec

Results :

Tests run: 7, Failures: 0, Errors: 0, Skipped: 0
```

Aplicación desplegada correctamente en el entorno de producción:

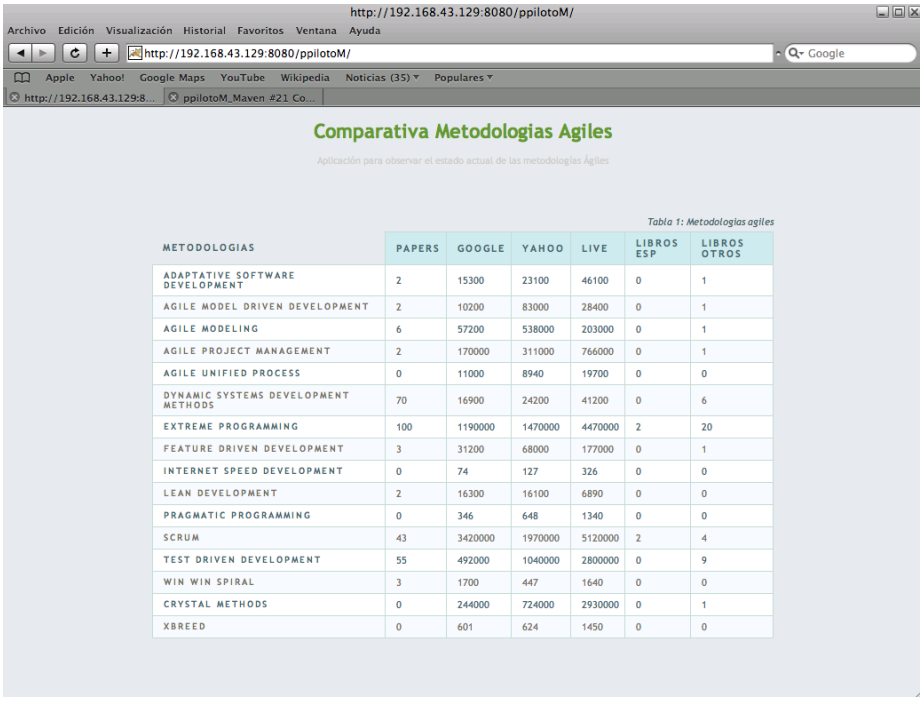


Tabla 1: Metodologias agiles

METODOLOGIAS	PAPERS	GOOGLE	YAHOO	LIVE	LIBROS ESP	LIBROS OTROS
ADAPTATIVE SOFTWARE DEVELOPMENT	2	15300	23100	46100	0	1
AGILE MODEL DRIVEN DEVELOPMENT	2	10200	83000	28400	0	1
AGILE MODELING	6	57200	538000	203000	0	1
AGILE PROJECT MANAGEMENT	2	170000	311000	766000	0	1
AGILE UNIFIED PROCESS	0	11000	8940	19700	0	0
DYNAMIC SYSTEMS DEVELOPMENT METHODS	70	16900	24200	41200	0	6
EXTREME PROGRAMMING	100	1190000	1470000	4470000	2	20
FEATURE DRIVEN DEVELOPMENT	3	31200	68000	177000	0	1
INTERNET SPEED DEVELOPMENT	0	74	127	326	0	0
LEAN DEVELOPMENT	2	16300	16100	6890	0	0
PRAGMATIC PROGRAMMING	0	346	648	1340	0	0
SCRUM	43	3420000	1970000	5120000	2	4
TEST DRIVEN DEVELOPMENT	55	492000	1040000	2800000	0	9
WIN WIN SPIRAL	3	1700	447	1640	0	0
CRYSTAL METHODS	0	244000	724000	2930000	0	1
XBREED	0	601	624	1450	0	0

2-Problemas e impedimentos del segundo Sprint

Dedicación parcial o inferior. Imprevistos temporales.

Necesidad de realizar las tareas administrativas (generación de pila de producto, tareas, etc …), consume mucho tiempo de desarrollo. Destacamos la importancia de tener roles dedicados.

Configuración del entorno de desarrollo, muy complicado trabajar con dos máquinas virtuales en un mismo equipo y con tan pocos recursos disponibles. Ha provocado una carga de trabajo y estrés asociado innecesarios en condiciones normales.

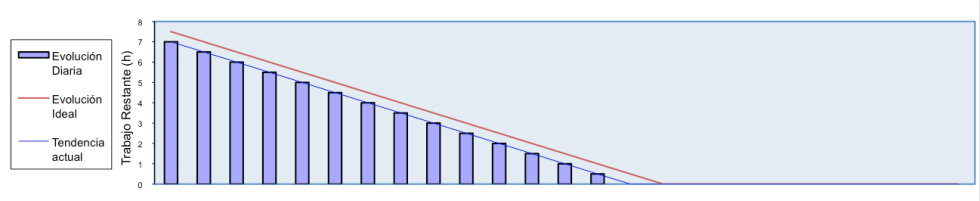
3-Revisión de la velocidad del Sprint 2

La velocidad del segundo Sprint ha sido de 7,5 unidades de historia o días ideales.

Sprint 2 Backlog

INICIO 13/06/2008
FIN 03/07/2008

Sprint implementation days	15			Effort	Remaining on implementation day...													
Trend calculated based on last	6	Days	Totals	7,5	7	6,5	6	5,5	5	4,5	4	3,5	3	2,5	2	1,5	1	0,5	
Task name	Story ID	Responsible	Status	Est.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Preparación entorno - IC, Subversion,...	2	Jose Carlos	Done	2	2	2	2	2	1,5	1	0,5	0	0	0	0	0	0	0	0
Formación JSP y servlets	2	Jose Carlos	Done	2	1,5	1	0,5	0	0	0	0	0	0	0	0	0	0	0	0
Diseñar Arquitectura de la aplicación	2	Jose Carlos	Done	1	1	1	1	1	1	1	1	1	0,5	0	0	0	0	0	0
Definir y escribir pruebas	2	Jose Carlos	Done	1	1	1	1	1	1	1	1	1	1	1	0,5	0	0	0	0
Obtener datos comparativa de la BBDD	2	Jose Carlos	Done	1,5	1,5	1,5	1,5	1,5	1,5	1,5	1,5	1,5	1,5	1,5	1,5	1,5	1	0,5	0



Observamos que la tarea de preparación del entorno de desarrollo ha sido añadida a este sprint y era una de las que tenía una estimación mayor. Ha sido un sprint muy disciplinado, que ha evolucionado muy bien, a pesar de encontrar algunas tareas más tediosas que otras.

4-Planificación del tercer Sprint

A continuación especificamos parámetros del sprint:

Fijamos una duración del sprint de 15 días laborables (3 semanas).

Estimación de la velocidad mediante un cálculo de recursos:

Equipo de 1 persona, disponible al 75%.

Días Disponibles

Jose Carlos 15 -2 días de fiesta

Total = 13 días-hombre disponibles

La velocidad estimada es inferior a 13, veamos cuanto inferior, utilizamos "factor de dedicación":

(DÍAS-HOMBRE DISPONIBLE) X (FACTOR DE DEDICACIÓN) = VELOCIDAD ESTIMADA

13 x 0.75 = 9.75 (velocidad estimada)

Podemos incrementar la dedicación de tiempo al proyecto y por tanto la velocidad también se ve alterada. Como la velocidad de la anterior iteración fue de 7,5 y obtenemos ahora una de 9,75 calculada, estimamos que podremos desarrollar una velocidad de 9.

Selección de las historias del product backlog:

PILA DE PRODUCTO:

Item Id	Nombre de la historia	Estado	Estimación	Sprint	Prioridad	Comentarios
1	Diseño apariencia de la aplicación.	Hecho	8	1	1	Modelo conceptual de la BBDD e insertar datos.
2	Mostrar comparativa del estudio de Metodologías.	Plan.	6	2	2	Utilización de servlets, jsp. Contenido dinámico
3	Login de usuario.	Plan.	3	3	3	
4	Añadir nuevas metodologías al estudio.	Plan.	3	3	4	Verificar campos obligatorios y formato de los datos introducidos.
5	Eliminar metodologías y todos sus datos.	Plan.	2	3	5	
6	Modificación datos introducidos.	Plan.	3	-	6	
7	Añadir papers como resultado de una búsqueda.	Plan.	6	-	7	Como se suben ficheros al servidor.
8	Consultar palabras clave de las búsquedas.	Plan.	1	-	8	
9	Top 5 metodologías ágiles.	Plan.	5	-	9	
10	Realizar automáticamente búsquedas y actualizar los datos.	Plan.	10	-	10	
11	Generar gráficos mejor documentación y mayor presencia en Internet.	Plan.	10	-	11	

No se modifican los ítems de la pila de producto. Seleccionamos los ítems que nos permiten nuestra velocidad.

Planificación del Sprint 3, tareas:

Item Id:	**3**	Item Backlog:	Login de usuario		
Tarea:	Modificar diseño de la página al añadir login.				
Responsable: Jose Carlos					
Estimación inicial		Trabajo hecho	0	Trabajo restante	1

Item Id:	**3,4,5**	Item Backlog:	-		
Tarea:	Implementar llamadas base de datos.				
Responsable: Jose Carlos					
Estimación inicial	**4**	Trabajo hecho	0	Trabajo restante	4

Item Id:	**3,4,5**	Item Backlog:	-		
Tarea:	Definir y escribir pruebas				
Responsable: Jose Carlos					
Estimación inicial	**1**	Trabajo hecho	0	Trabajo restante	1

Story ID:	**3,4,5**	Item Backlog:	-		
Tarea:	Diseñar botones y comportamiento.				
Responsable: Jose Carlos					
Estimación inicial	**2**	Trabajo hecho	0	Trabajo restante	2

Resumen de los detalles discutidos y/o acuerdos alcanzados en la reunión

Presentación de la demo del producto, pruebas realizadas y entornos de desarrollo.

Análisis del gráfico burn-down, determina que es necesario un incremento de la velocidad para alcanzar los objetivos. La velocidad debe ser de 9 puntos de historia.

Planificación del tercer Sprint y sus tareas.

Asignación de tareas:

Jose Carlos es Product Owner.

Jose Carlos es Scrum Master.

Jose Carlos es equipo de desarrollo.

Planificación de los puntos a discutir y/o que deben realizarse para la siguiente reunión:

Presentación de una versión funcional del producto con los requisitos mínimos establecidos ya implementados y probados.

Revisión y análisis de la velocidad y productividad del sprint.

Señalar problemas e impedimentos en el tercer Sprint.

Establecer la necesidad de implementar más funcionalidades o por el contrario dar el proyecto por finalizado.

4. *Acta de la reunión final del tercer sprint y retrospectiva del proyecto.*

Piloto Metodologías Ágiles

Anotaciones de la reunión

Equipo: 46 Día: 26/07/08 Inicio/Fin hora: 08h30/14h Oficinas de Everis Diagonal

Asistentes:

Jose Carlos Carvajal (Scrum Master y Product Owner)

Jose Carlos Carvajal (Equipo de desarrolladores.)

Ausentes:

No hay ausentes.

Temas a tratar en la reunión:

1-Demo del Sprint3

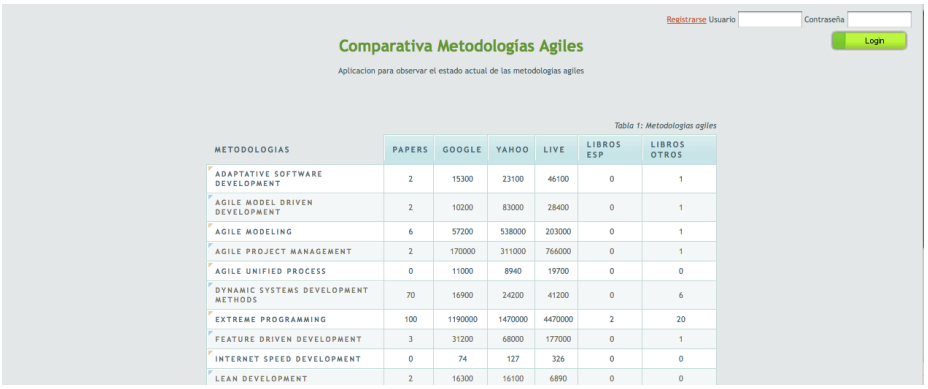
En esta demo mostramos que todas las historias requeridas han sido satisfechas y mostramos las pruebas realizadas a la aplicación.

Historia 1 - Diseño apariencia de la aplicación. Esta historia fue realizada en el primer sprint.

Historia 2 - Mostrar comparativa del estudio de Metodologías. Historia realizada en el segundo sprint.

Historia 3- Login de usuario.

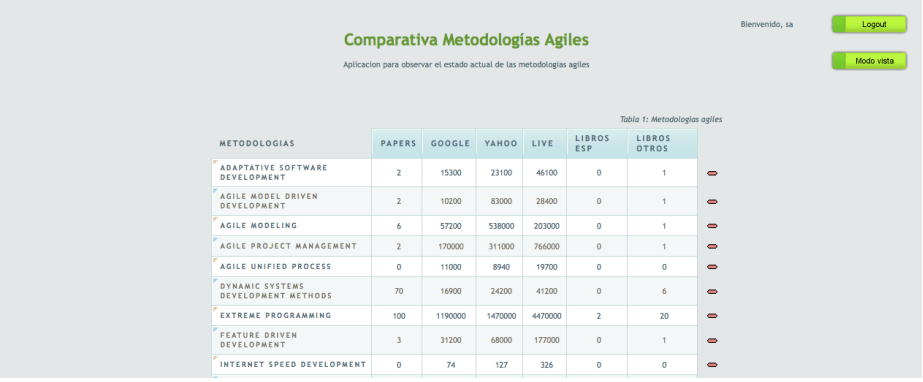
A continuación mostramos capturas de pantalla que nos permiten verificar esta historia.



METODOLOGIAS	PAPERS	GOOGLE	YAHOO	LIVE	LIBROS ESP	LIBROS OTROS
ADAPTATIVE SOFTWARE DEVELOPMENT	2	15300	23100	46100	0	1
AGILE MODEL DRIVEN DEVELOPMENT	2	10200	83000	28400	0	1
AGILE MODELING	6	57200	538000	203000	0	1
AGILE PROJECT MANAGEMENT	2	170000	311000	766000	0	1
AGILE UNIFIED PROCESS	0	11000	8940	19700	0	0
DYNAMIC SYSTEMS DEVELOPMENT METHODS	70	16900	24200	41200	0	6
EXTREME PROGRAMMING	100	1190000	1470000	4470000	2	20
FEATURE DRIVEN DEVELOPMENT	3	31200	68000	177000	0	1
INTERNET SPEED DEVELOPMENT	0	74	127	326	0	0
LEAN DEVELOPMENT	2	16300	16100	6890	0	0

fig 1

Podemos observar en la figura 1 que se ha añadido la opción de autenticarse y ademas registrarse como nuevo usuario. En la figura 2 observamos como cambia el aspecto de la página principal cuando un usuario se autentica y activa el modo edición.



METODOLOGIAS	PAPERS	GOOGLE	YAHOO	LIVE	LIBROS ESP	LIBROS OTROS
ADAPTATIVE SOFTWARE DEVELOPMENT	2	15300	23100	46100	0	1
AGILE MODEL DRIVEN DEVELOPMENT	2	10200	83000	28400	0	1
AGILE MODELING	6	57200	538000	203000	0	1
AGILE PROJECT MANAGEMENT	2	170000	311000	766000	0	1
AGILE UNIFIED PROCESS	0	11000	8940	19700	0	0
DYNAMIC SYSTEMS DEVELOPMENT METHODS	70	16900	24200	41200	0	6
EXTREME PROGRAMMING	100	1190000	1470000	4470000	2	20
FEATURE DRIVEN DEVELOPMENT	3	31200	68000	177000	0	1
INTERNET SPEED DEVELOPMENT	0	74	127	326	0	0

fig 2

A continuación, en la figura 3, mostramos la posibilidad que nos aparece, cuando estamos autenticados, de crear nuevas metodologías.

AGILE MODELING	6	57200	538000	203000	0	1
AGILE PROJECT MANAGEMENT	2	170000	311000	766000	0	1
AGILE UNIFIED PROCESS	0	11000	8940	19700	0	0
DYNAMIC SYSTEMS DEVELOPMENT METHODS	70	16900	24200	41200	0	6
EXTREME PROGRAMMING	100	1190000	1470000	4470000	2	20
FEATURE DRIVEN DEVELOPMENT	3	31200	68000	177000	0	1
INTERNET SPEED DEVELOPMENT	0	74	127	326	0	0
LEAN DEVELOPMENT	2	16300	16100	6890	0	0
PRAGMATIC PROGRAMMING	0	346	648	1340	0	0
SCRUM	43	3420000	1970000	5120000	2	4
TEST DRIVEN DEVELOPMENT	55	492000	1040000	2800000	0	9
WIN WIN SPIRAL	3	1700	447	1640	0	0
COM+	0	100	500	200	0	0
CRYSTAL METHODS	0	244000	724000	2930000	0	1
XBREED	0	601	624	1450	0	0

fig 3

Historias 4 y 5- Añadir nuevas metodologías al estudio y eliminar metodologías y todos sus datos.

Si queremos crear una nueva metodología, tan solo hemos de clicar en el botón añadir y rellenar el formulario que observamos en al figura 4. En caso de querer eliminar una de las metodologías existente, hemos de clicar en el simbolo negativo de color rojo que aparece al lado izquierdo de cada metodología (ver figura 5).

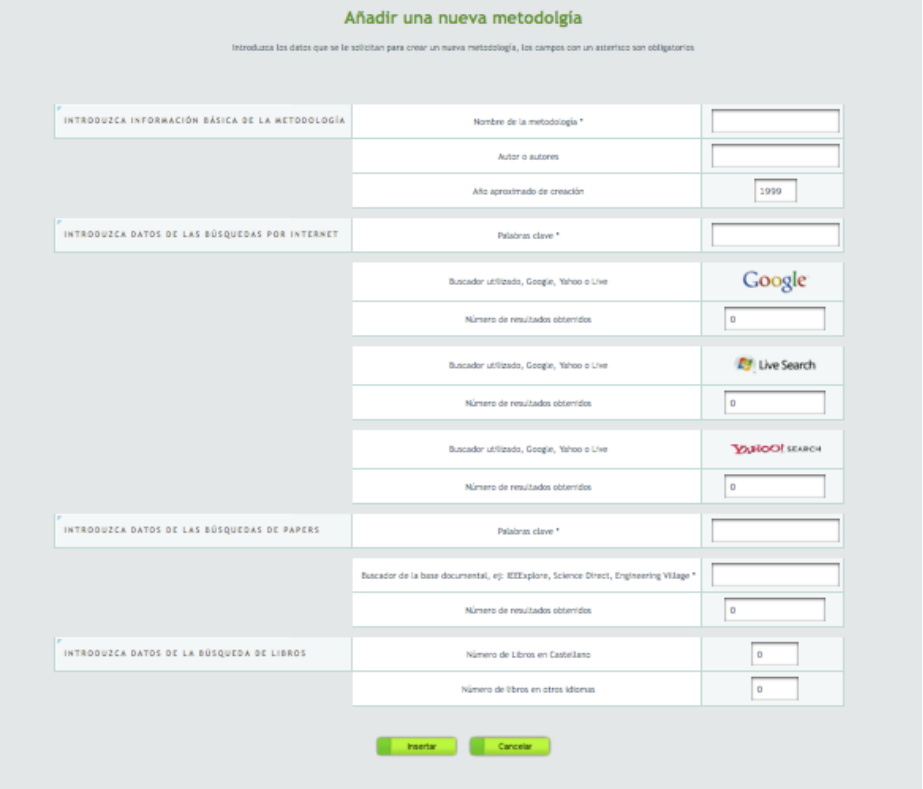


fig 4

Tabla 1: Metodologías agiles

	LIBROS ESP	LIBROS OTROS	
00	0	1	
00	0	1	
000	0	1	

fig 5

Una opción no requerida por el usuario era la de crear nuevos usuarios, pero hemos considerado este requisito implicito al login. Para crear un nuevo usuario, clicar el vínculo registrar de la página principal y rellenar el formulario de la figura 6.

2-Problemas e impedimentos del tercer Sprint

A pesar de ser uno de los sprint con mayor número de historias para desarrollar, no ha supuesto un cambio significativo con respecto a los otros. Todas las tareas has podido ser finalizadas sin mayores problemas.

Destacamos la problemática de tener que autogestionarse al mismo tiempo que se lleva a cabo un proyecto, es difícil ser estricto y no adecuar el plan a tus necesidades individuales.

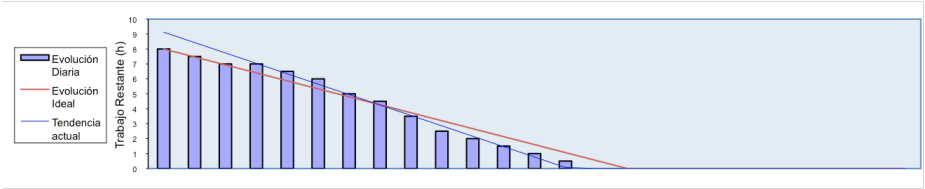
3-Revisión de la velocidad del Sprint 3

La velocidad del tercer Sprint ha sido de 8 puntos de historia o hombres-día ideales.

Sprint 3 Backlog

INICIO 07/07/2008
FIN 25/07/2008

Sprint implementation days	15			Effort	Remaining on implementation day...														
Trend calculated based on last	7,5	Days	Totals	8	8	7,5	7	7	6,5	6	5	4,5	3,5	2,5	2	1,5	1	0,5	
Task name	Story ID	Responsible	Status	Est.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Definir y escribir pruebas.	3,4,5	Jose Carlos	Done	1	1	0,5	0	0	0	0	0	0	0	0	0	0	0	0	0
Modificar diseño de la página al añadir login.	3	Jose Carlos	Done	1	1	1	1	1	0,5	0	0	0	0	0	0	0	0	0	0
Diseñar botones y comportamiento.	3,4,5	Jose Carlos	Done	2	2	2	2	2	2	2	1	0,5	0	0	0	0	0	0	0
Implementar llamadas a BBDD.	3,4,5	Jose Carlos	Done	4	4	4	4	4	4	4	4	4	3,5	2,5	2	1,5	1	0,5	0



Como podemos observar en el diagrama de progreso o burn-down, la evolución del proyecto inicialmente fue muy lenta (hubo dos días que no se avanzo) y parecía que el ritmo era insuficiente, pero al poder dedicar mayor tiempo y recursos al proyecto, el ritmo aumento y todas las tareas se finalizaron a tiempo.

4- Necesidad de un cuarto sprint.

Teniendo en cuenta la prioridad de las historias, tan solo hay un historia que esta sin implementar y que esta catalogada como que debería implementarse.
Consideramos que las historias implementadas son suficientes para dar el proyecto por finalizado.

Resumen de los detalles discutidos y/o acuerdos alcanzados en la reunión

Presentación de la demo del producto y pruebas realizadas.

Análisis del gráfico burn-down, determina que hubo una concentración de trabajo en los últimos días del sprint. La velocidad fue de 8 puntos de historia.

Se da el proyecto piloto por concluido.

Asignación de tareas:

Jose Carlos es Product Owner.

Jose Carlos es Scrum Master.

Jose Carlos es equipo de desarrollo.